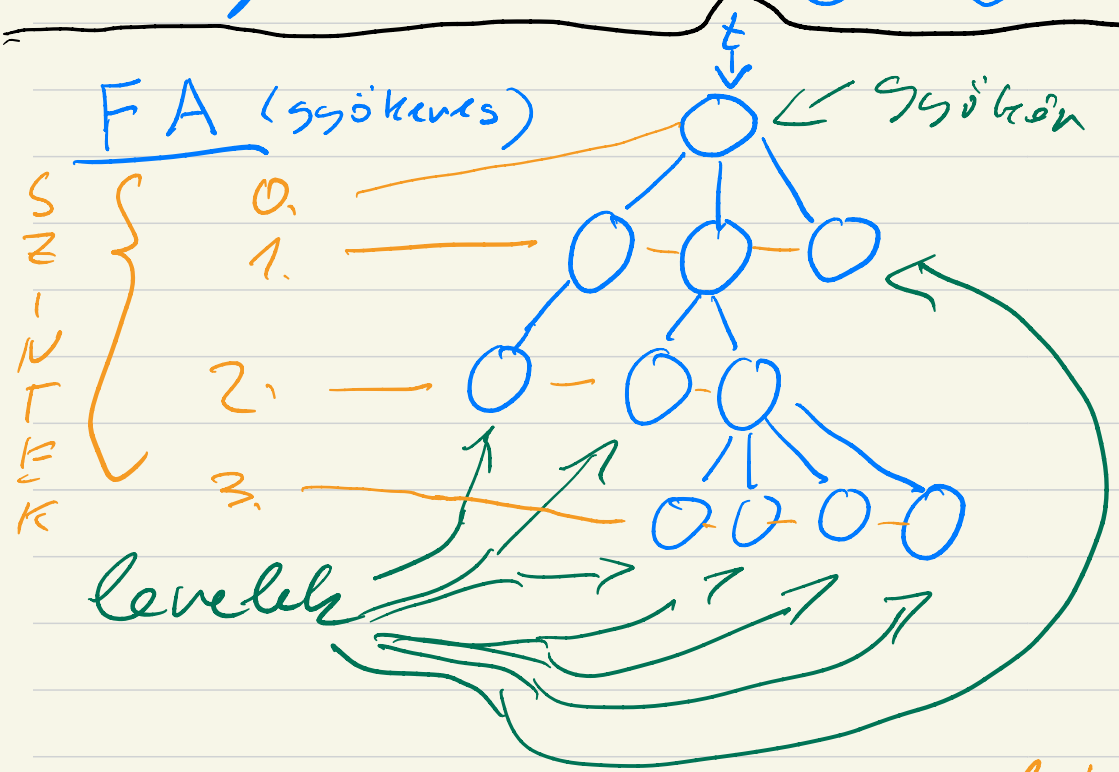
Eddig LIN. ASZ.:

FA (gyökeres)



$h(t) = 3$

belső csúcs: 1 levél

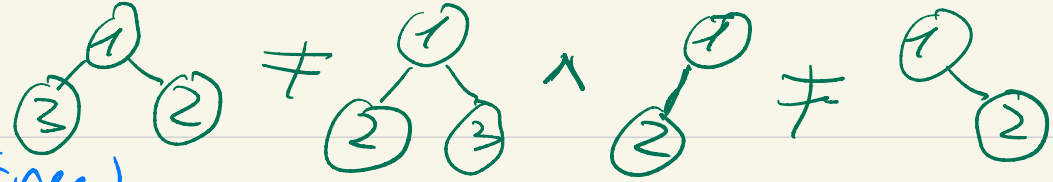
$h(\overset{\downarrow}{\circ}) = 0$

$\overset{\downarrow}{\circ}$ { gyökér / levél

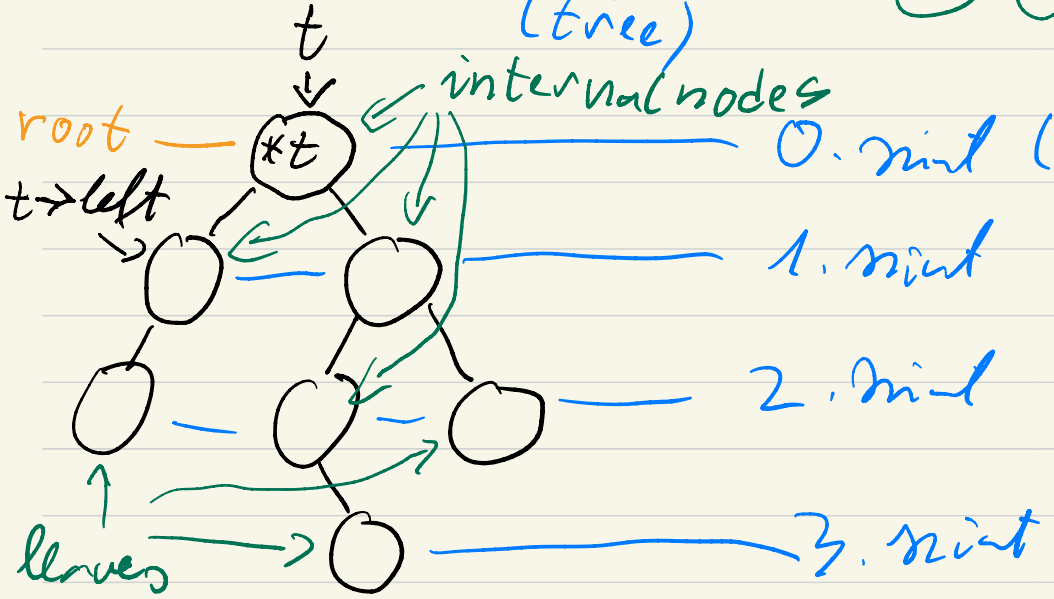
Ω: üres fa (nincs csúcsa)

$h(\Omega) \overset{def}{=} -1$

Bináris fák



(tree)

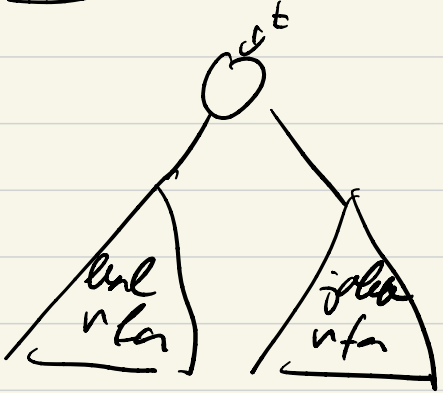


bináris fa
(binary tree)

$h\left(\overset{\downarrow}{\bigcirc}\right) = 0$ $\qquad h(\Omega) \overset{def}{=} -1$

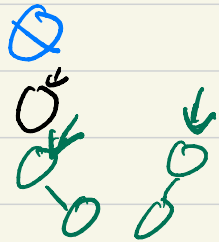
$t \neq \Omega \Rightarrow h(t) = 1 + \max(h(t\text{->left}), h(t\text{->right}))$

Részfák:

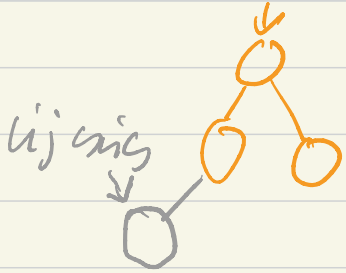


részfa (subtree)

reflexív } relációó
tranzitív

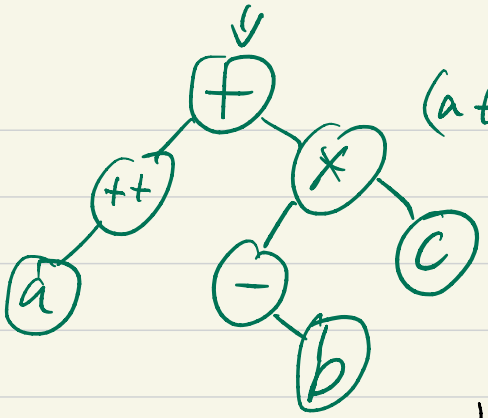


csúcsok	üres rfák	rfák
0	1	1
1	2	3
2	3	5
3	4	7
4	5	9



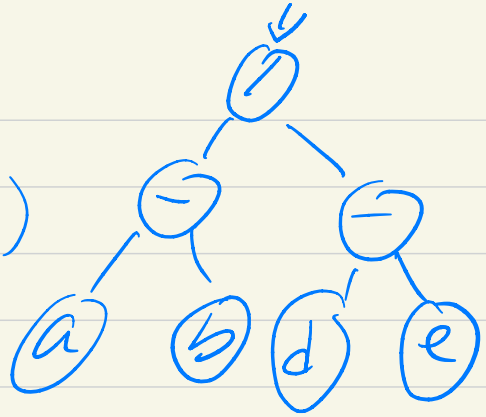
üres rfa:
$-1+2=+1$

n csúcs → n üres rfa } (2n+1) db rfa
↓
(n+1) üres rfa

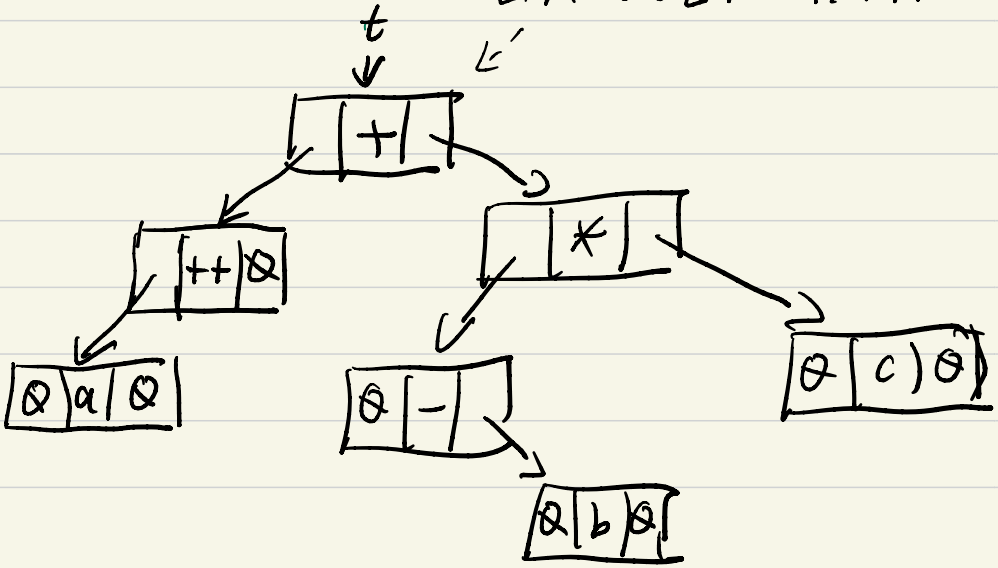


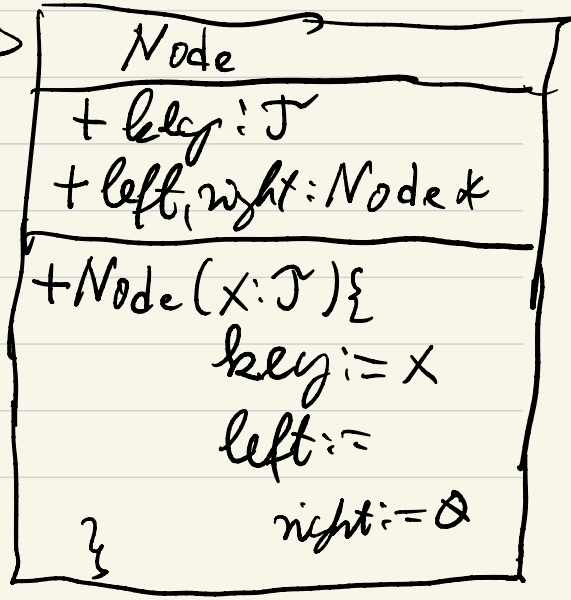
(a++) + (-b) * c

(a-b)/(d-e)

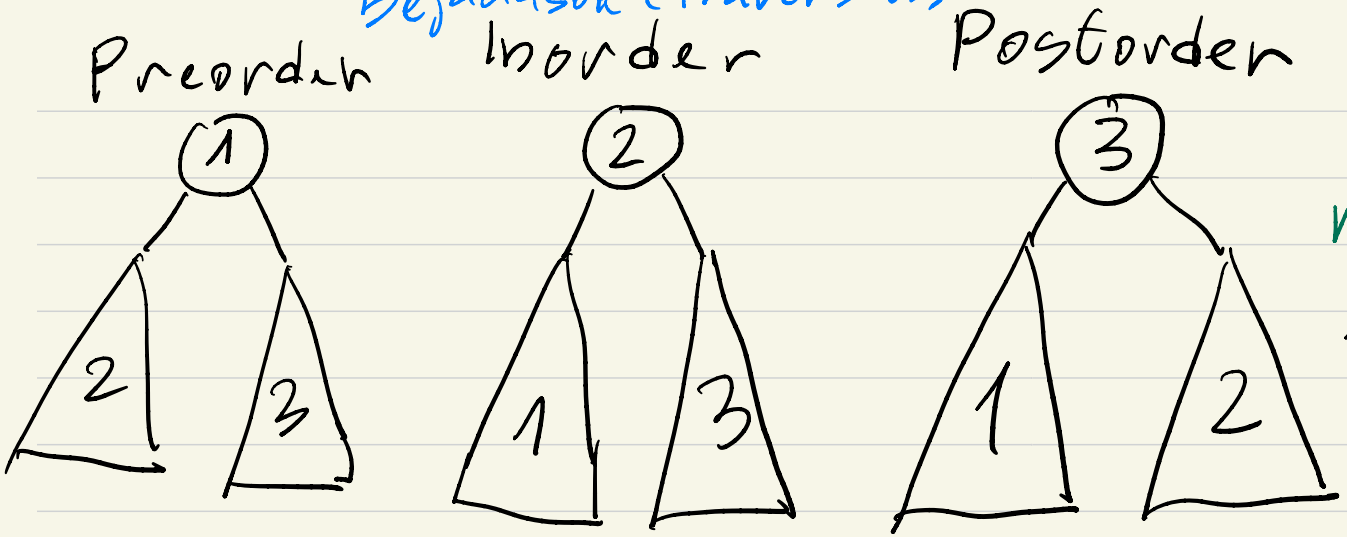


LÁNCOLT ÁBRÁZOLÁS





Bejárások (Traversals)



$n := |t| = n(t)$

A t fa csúcsainak száma.

```
Inorder(t: Node*)
    t ≠ ∅
        Inorder(t->left)
        process(t)
        Inorder(t->right)
```

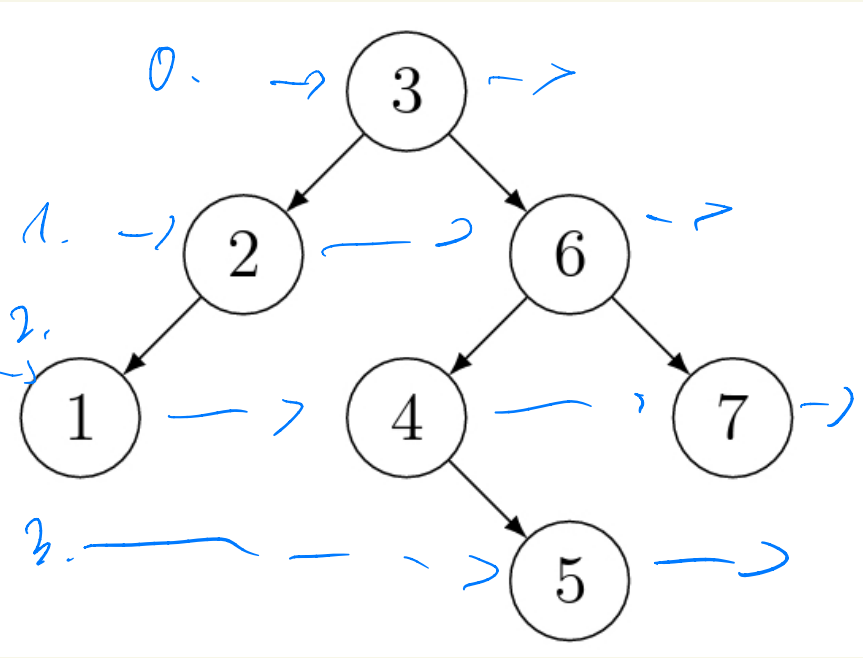
preorder: process(t) az első

postorder: process(t) az utolsó

$T(n) \in \Theta(n)$

[n csúcs $\Rightarrow 2n+1$ részfa]

Level-order traversal (szintenkénti bejárás):



A Q sort használjuk.

Kezdetben:

$Q = \langle 3 \rangle$ — 0. szint

Ciklusban kivesszük Q első elemét, feldolgozzuk, majd a gyerekeit Q végére tesszük.

$Q = \langle 2, 6 \rangle$ — 1. szint

$Q = \langle 6, 1 \rangle$

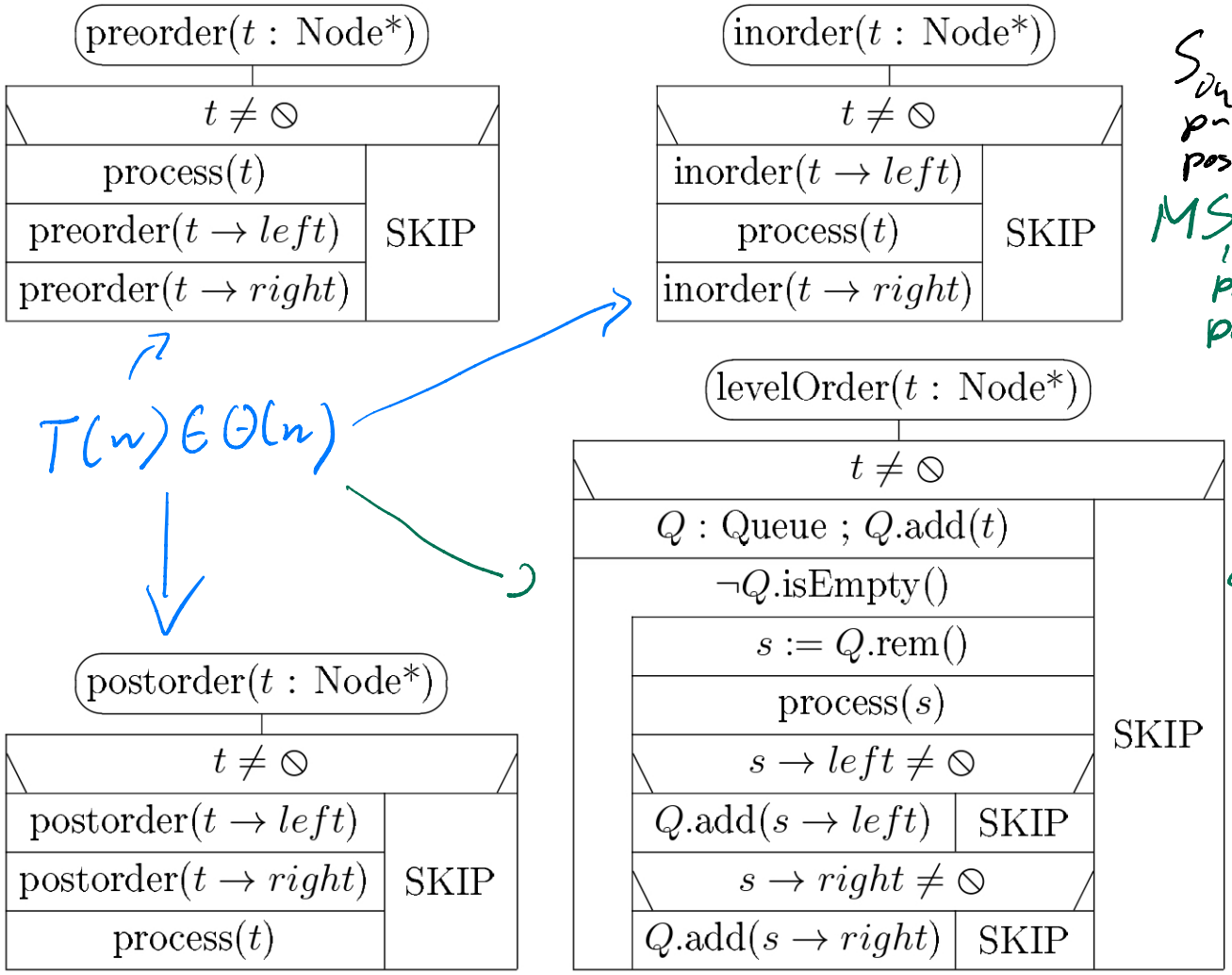
$Q = \langle 1, 4, 7 \rangle$ — 2. szint

$Q = \langle 4, 7 \rangle$

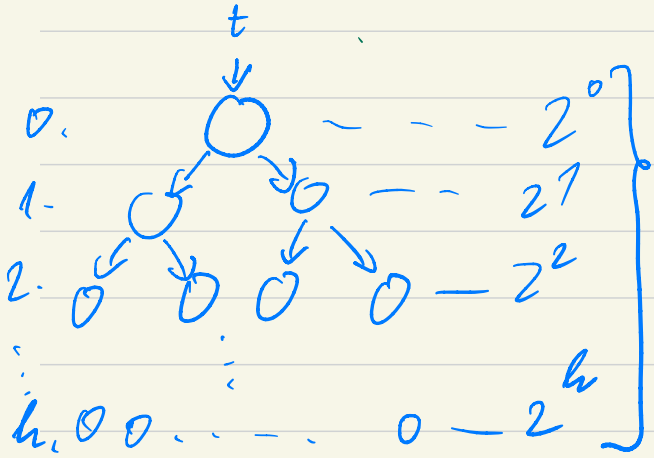
$Q = \langle 7, 5 \rangle$

$Q = \langle 5 \rangle$ — 3. szint

$Q = \langle \rangle$ STOP!


preorder(t : Node*)
t ≠ ⊘
process(t)
preorder(t → left)
preorder(t → right)
SKIP
inorder(t : Node*)
t ≠ ⊘
inorder(t → left)
process(t)
inorder(t → right)
SKIP
levelOrder(t : Node*)
t ≠ ⊘
Q : Queue ; Q.add(t)
¬Q.isEmpty()
s := Q.rem()
process(s)
s → left ≠ ⊘
Q.add(s → left)
SKIP
s → right ≠ ⊘
Q.add(s → right)
SKIP
SKIP
postorder(t : Node*)
t ≠ ⊘
postorder(t → left)
postorder(t → right)
process(t)
SKIP
T(n) ∈ Θ(n)
S_in (h) ∈ Θ(h)
in
pre
post
MS (n) ∈ Θ(n)
in
pre
post
mS_in (n) ∈ Θ(log n)
pre
post
it(n) = n
MS_level (n) ∈ Θ(n)
mS_level (n) ∈ Θ(1)

t lin fa: $h(t)$, $n(t) = |t|$



$$n = \sum_{i=0}^{h} 2^i = 2^{h+1} - 1$$

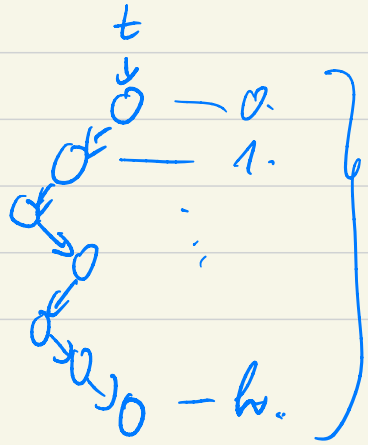
tétel:
$n < 2^{h+1}$

$n > 0 \Rightarrow \log n < h+1$

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

lin. fa.

$n = h+1$



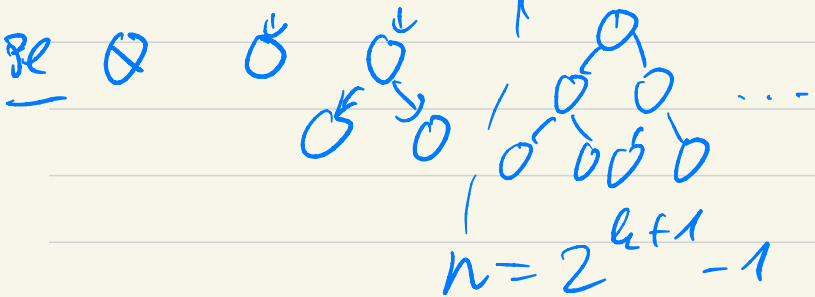
tétel:
$n \geq h+1$

D: t bin fa: szig. bin. (full) $\overset{def}{\iff}$ ∀ belső csúcsnak 2 gyereke ∃.

$\overset{Tul}{\Leftarrow}$ $\ell(t) = i(t)+1$ $(t \neq \Lambda)$

$\Rightarrow$ MergeSort(A:T[n]) $\leftarrow (2n-1)$ ms() hívás

D: t bin. fa tökéletes $\overset{def}{\iff}$ szig. bin. ∧ ∀ levele azonos mélységben van.

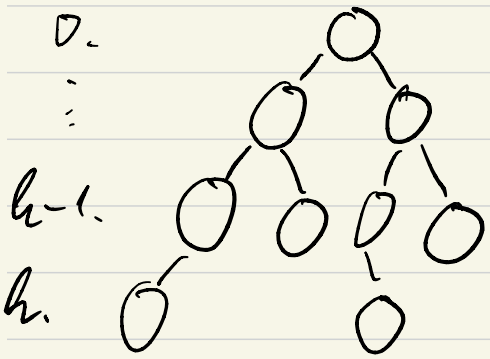
Pl 

$n = 2^{\ell+1} - 1$

∀ tökéletes bin. fa majdnem teljes.

D: t. bin. fa majdnem teljes (nearly complete) $\overset{def}{\iff}$ ∃ legalsó szinten levő leveleket elvéve, a maradék fa tökéletes, vagy $t = \Lambda$

$h \geq n$



$0..h-1$. szintek $\rightsquigarrow$ h. szint, $1 \leq k \leq 2^h$

$$n = 2^h - 1 + k$$

$$2^h \leq n \leq 2^{h+1} - 1 < 2^{h+1} \quad (n > 0)$$

$$h \leq \log n < h+1$$

$$h \leq \lfloor \log n \rfloor < h+1$$

$$h = \lfloor \log n \rfloor \quad (n > 0)$$

Ti. t bin. fa magjában teljes $\wedge$ $n = |t| > 0$ $\wedge$ $h = h(t)$

$$\Longrightarrow h = \lfloor \log n \rfloor$$

$\Longleftarrow$ ↖ HF a logikailag ellenpélda?

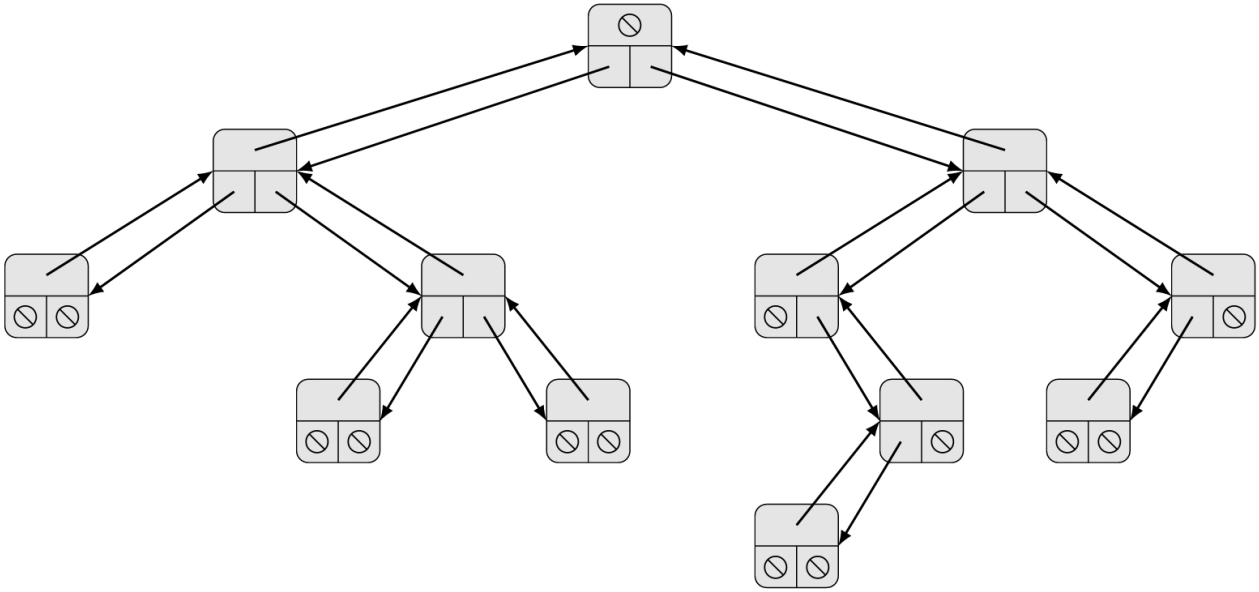
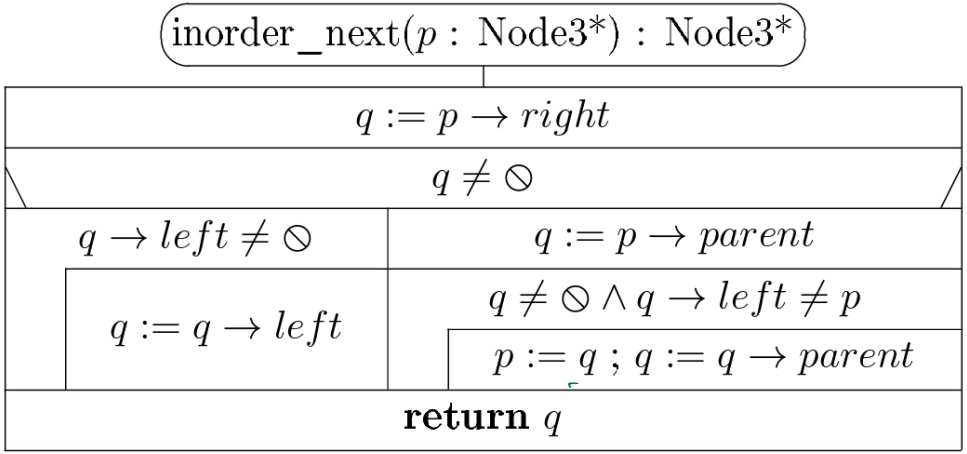


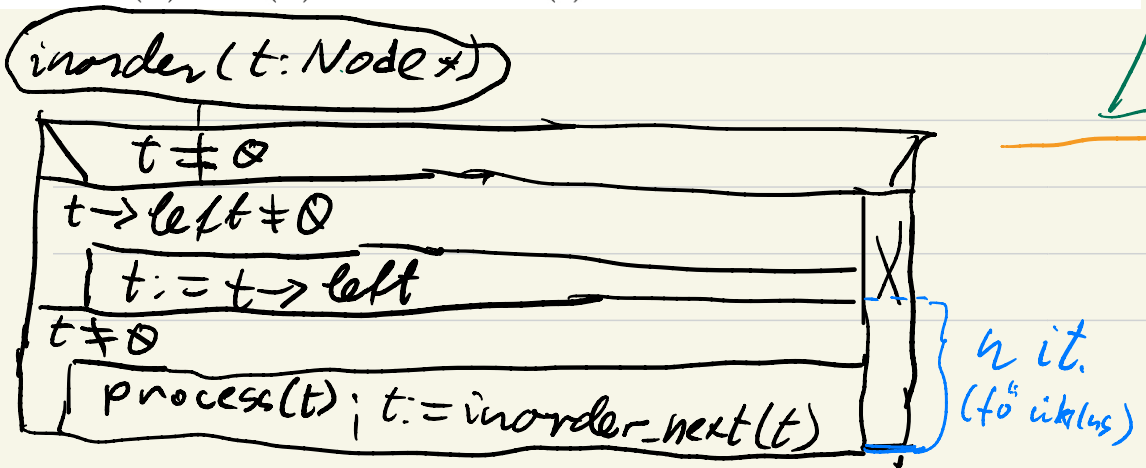
Figure 9: Binary tree with parent pointers. (We omitted the keys of the nodes here.)

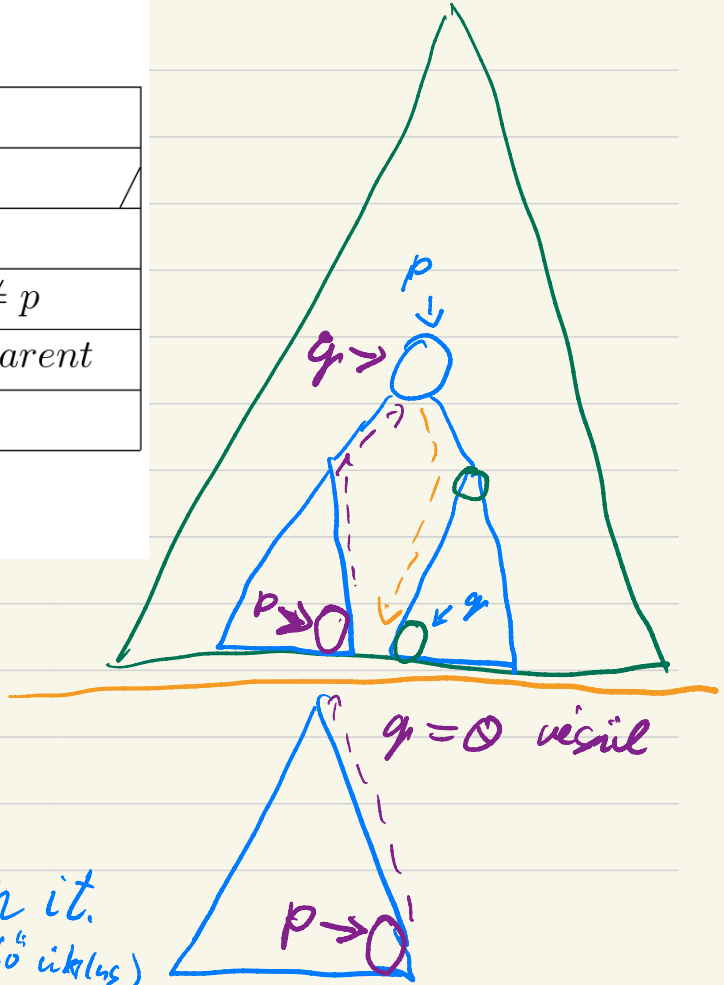
Node3
+ $key : \mathcal{T}$ // $\mathcal{T}$ is some known type + $left, right, parent$: Node3*
+ Node3(p:Node3*) { $left := right := \oslash$; $parent := p$ } + Node3($x : \mathcal{T}$, p:Node3*) { $left := right := \oslash$; $parent := p$; $key := x$ }

6.4.2 Using parent pointers



$MT(h) \in O(h)$ where $h = h(t)$





Δ $q := p \to right ; q \neq 0$
* $q := q \to left$
* $t := t \to left$

} megy oda, ahol még nem volt } $(n-1)$-szer

Δ $q := p \to parent$
* $q := q \to parent$

} végiglépked egy szinten } n-szer

elhagy

* ciklus it.-val (fő ciklus)

* $(2n-1)$ lépés

Δ inorder_next() : hívásához } : n db. =

= fő ciklus it. $= n$

ebből

$2n-1$ ciklus it.

$* \wedge \Delta \Rightarrow (n-1)$ db egyéb ciklus it. *

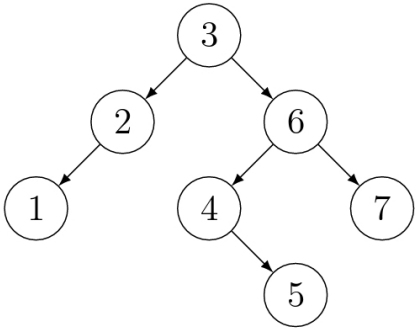
$T(n) = \Theta(n) = \Theta(2n-1)$

6.5 Parenthesised, i.e. textual form of binary trees

Parenthesised, i.e. the textual form of a nonempty binary tree:

(LeftSubtree Root RightSubtree)

A pair of brackets represents the empty tree. We omit the empty subtrees of the leaves. We can use different kinds of parentheses for easier reading. For example, see Figure 11.



The binary tree on the left in

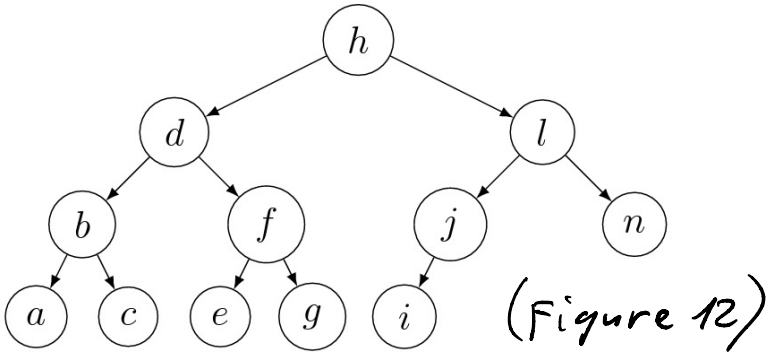
- simple textual form:
  (((1) 2 ()) 3 ((() 4 (5)) 6 (7)))
- elegant parenthesised form:
  { [(1) 2 ()] 3 [(⟨⟩ 4 ⟨5⟩) 6 (7)] }

Figure 11: The same binary tree in graphical and textual representations. Notice that by omitting the parenthesis from the textual form of a tree, we receive the inorder traversal of that tree.

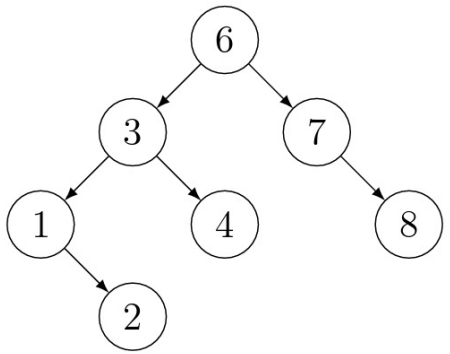
Bináris keresőfa (BST)

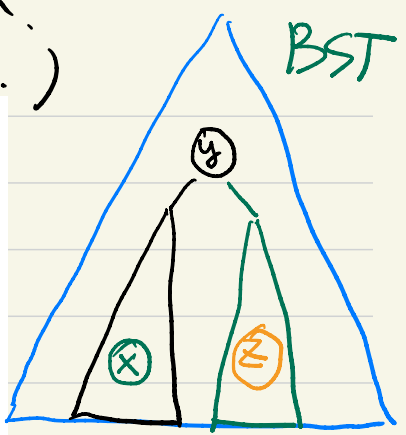
(Szótárak repr. műv.: szótár impl.)

A *binary search tree* (BST) is a binary tree whose nodes each store a *key* (and, optionally, some associated value). The tree additionally satisfies the binary search tree property, which states that the key in each node must be greater than any key stored in the left subtree and less than any key stored in the right subtree. (See Figure 12. Notice that there is no duplicated key in a BST.) A *binary sort tree* is similar to a BST but may have equal keys. The key in each node must be greater than or equal to any key stored in the left subtree and less than or equal to any key stored in the right subtree. In this Lecture Notes, all the subsequent structure diagrams refer to BSTs.



(Figure 12)

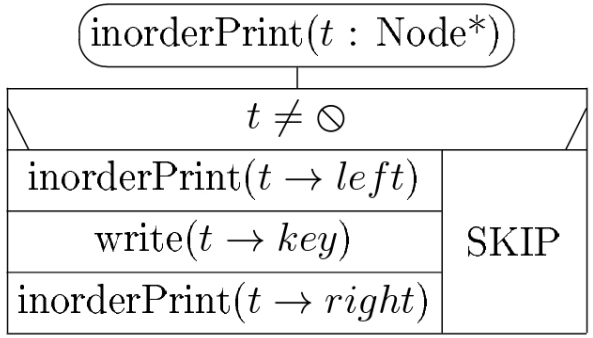




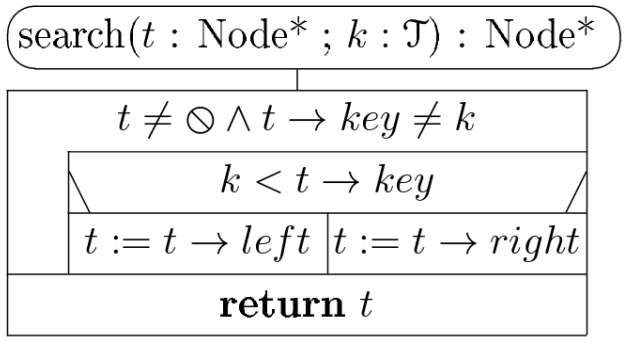
BSoT

$x \leq y$ $y \leq z$

bináris rendezőfa

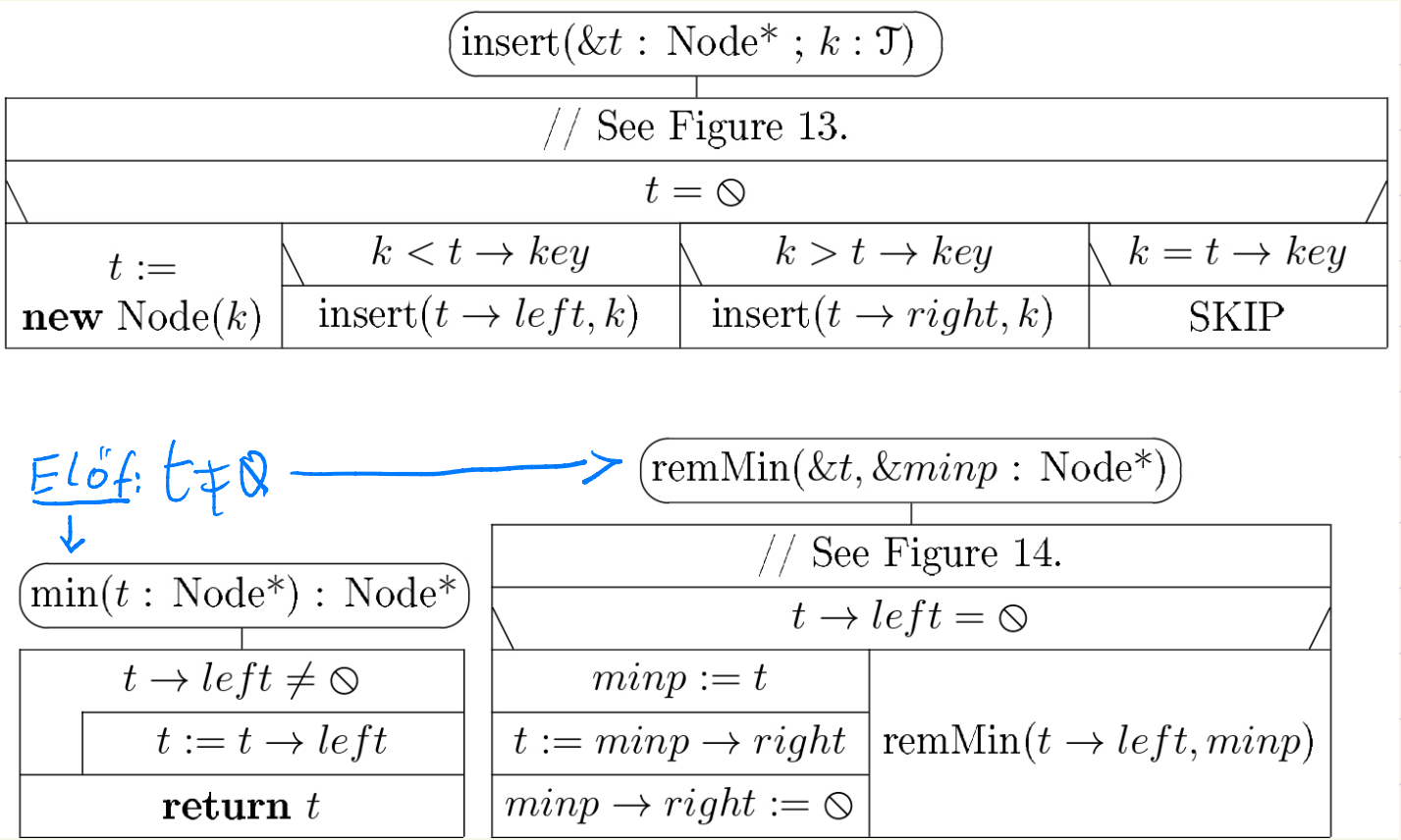


Property 6.7 *Procedure inorderPrint(t: Node*) prints the keys of the binary tree t in strictly increasing order,* $\iff$ *binary tree t is a search tree.*



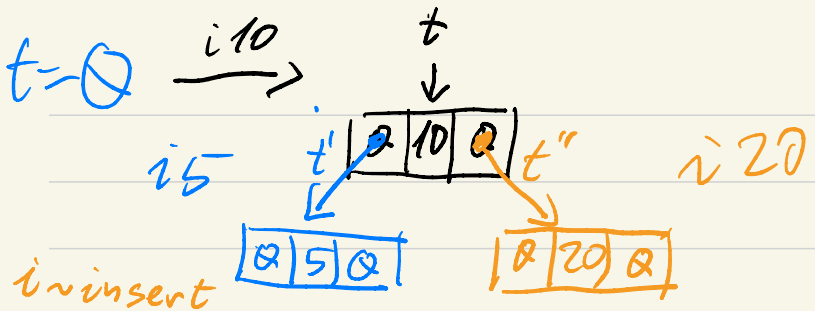
$MT(h) \in \Theta(h)$

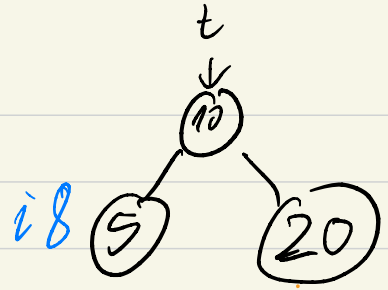
$MS(h) \in \Theta(1)$



MT_ins(h) ∈ Θ(h)
Search
min, remMin

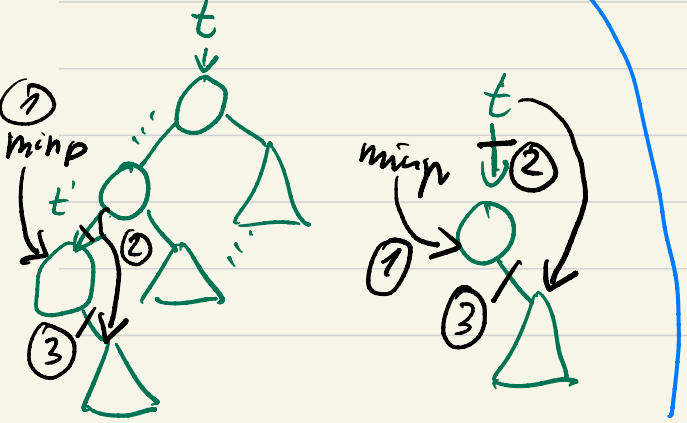
mT_ins(h) ∈ Θ(1)
Search
min, remMin





t' ~ t -> left t" ~ t -> right

remMin:



Véletlenszerű input => a beszúrások után a BST magasságának várható értéke $\approx 1{,}38 \cdot \log n$

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

search
insert
min, remMin
del

$MT(n) \in \Theta(n)$

search, insert
min, remMin, del

del(&t : Node* ; k : $\mathcal{T}$)

// See Figure 16.			
$t \neq \oslash$			
$k < t \to key$	$k > t \to key$	$k = t \to key$	SKIP
del($t \to left, k$)	del($t \to right, k$)	delRoot(t)	

delRoot(&t : Node*)

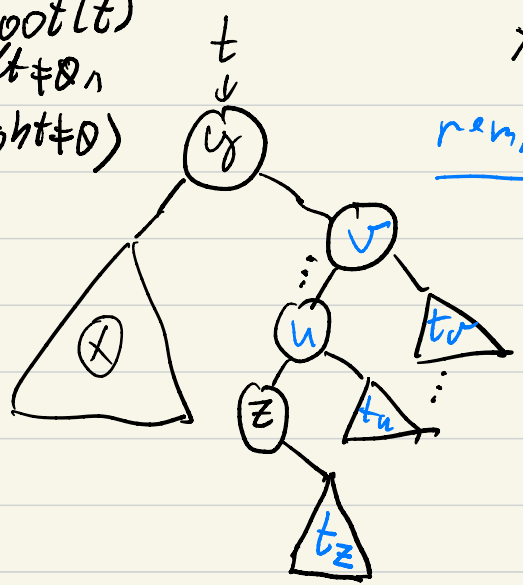
// See Figure 17.		
$t \to left = \oslash$	$t \to right = \oslash$	$t \to left \neq \oslash \wedge t \to right \neq \oslash$
$p := t$	$p := t$	remMin($t \to right, p$)
$t := p \to right$	$t := p \to left$	$p \to left := t \to left$; $p \to right := t \to right$
delete p	**delete** p	delete t ; $t := p$

$MT_{\text{search}}(h), MT_{\text{insert}}(h), MT_{\text{min}}(h) \in \Theta(h)$
$MT_{\text{remMin}}(h), MT_{\text{del}}(h), MT_{\text{delRoot}}(h) \in \Theta(h)$
where $h = h(t)$.

$MT_{search}(h)$
insert
min
remMin
del, delRoot
$\Downarrow$
(h)

MT_{search}
insert
min, remMin
del, delRoot

delRoot(t)
(t->left≠0 ∧
t->right≠0)



remMax(t)

$x < y < z <$ bármely z-től különböző kulcs a t->right rfában

KUPACOK, Prioritásos sorok

(Bináris fa)

D. Teljes fa a $t \overset{\text{def}}{\iff} t = \emptyset \vee t$ majdnem teljes $\wedge$ alsó szint jobbra tömörített.
(complete)

alapértelmezés

D. t bináris fa [maximum] kupac (heap) $\overset{\text{def}}{\iff}$
t teljes $\wedge$ $\forall$ szülő $\geq$ gyerekei

D. t bináris minimum kupac $\iff$ t teljes $\wedge$
$\forall$ szülő $\leq$ gyerekei

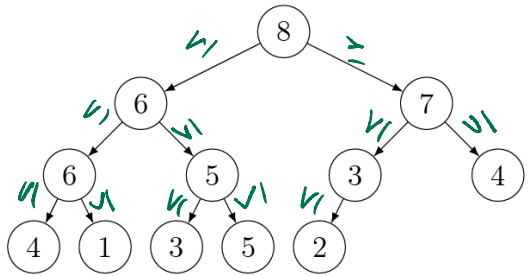


Figure 18: A *(binary maximum) heap*. It is a complete binary tree. The key of each parent is $\geq$ to the child's key.

$h = \lfloor \log n \rfloor$

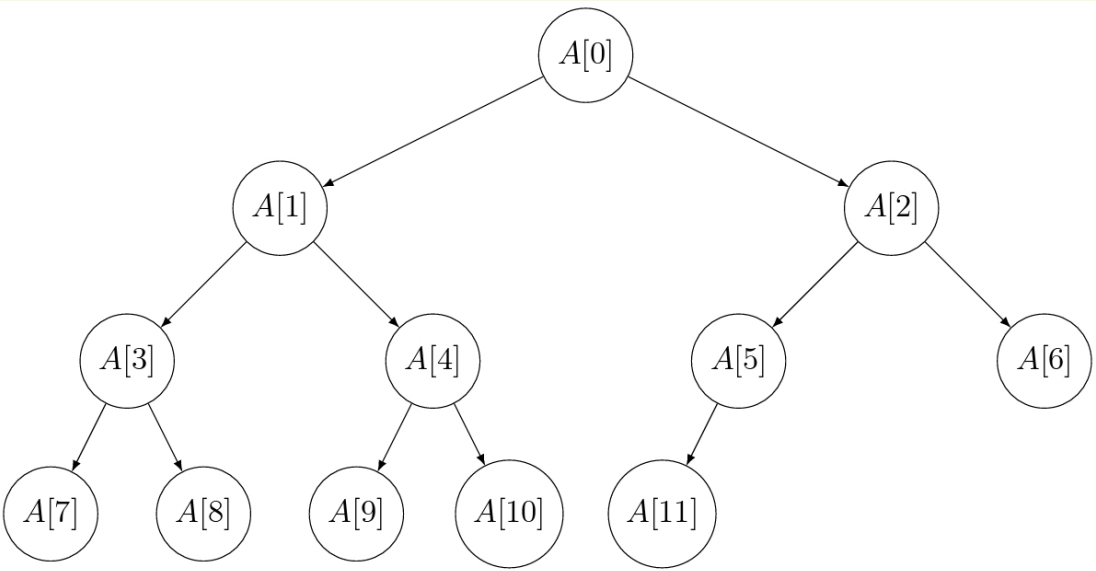
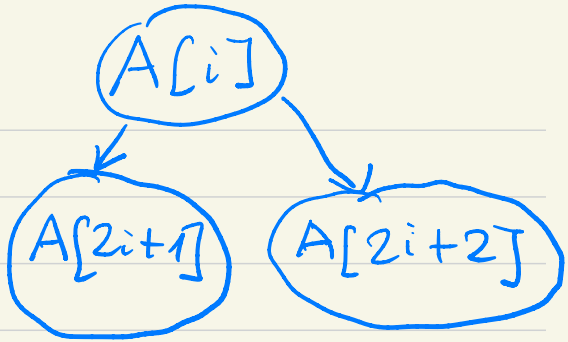


Figure 19: A complete binary tree of size 12 represented by the first 12 elements of an array A: We put the tree nodes into the array in level order.



$left(i) = 2i+1$

$right(i) = 2i+2$

$parent(j) = \left\lfloor \frac{j-1}{2} \right\rfloor$

$root(j) = (j = 0)$

$nonroot(j) = (j > 0)$

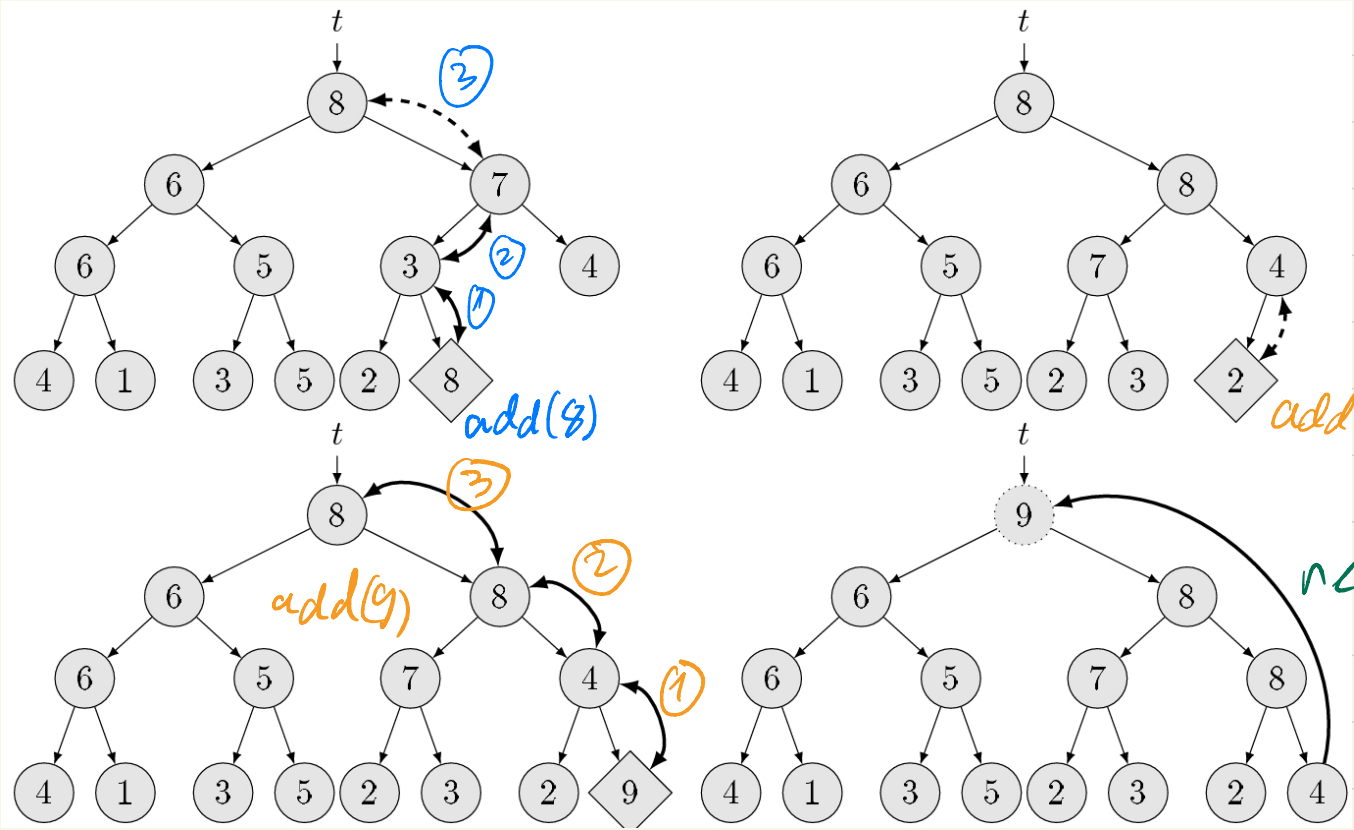
$internal(i) = (2i+1 < n)$

$leaf(i) = (2i+1 \geq n)$

PRIORITY Queue

PrQueue
− $A : \mathcal{T}[]$ // $\mathcal{T}$ is some known type ; $A.length$ is the physical
− constant $m0 : \mathbb{N}_+ := 16$ // size of the PrQ, its default is m0. $T(n) \in \Theta(1)$
− $n : \mathbb{N}$ // $n \in [0..A.length]$ is the actual length of the PrQ
+ PrQueue($m : \mathbb{N}_+ := m0$){ $A :=$ **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** $n = 0$}
+ $\sim$ PrQueue() { **delete** A }
+ setEmpty() {$n := 0$} // reinitialize the priority queue

$T(n) \in \Theta(1)$


t
8
6
7
6
5
3
4
4
1
3
5
2
8
3
2
1
add(8)
t
8
6
8
6
5
7
4
4
1
3
5
2
3
2
add(2)
t
8
6
8
6
5
7
4
4
1
3
5
2
3
2
9
add(9)
3
2
1
t
9
6
8
6
5
7
8
4
1
3
5
2
3
2
4
maxMax() = 9

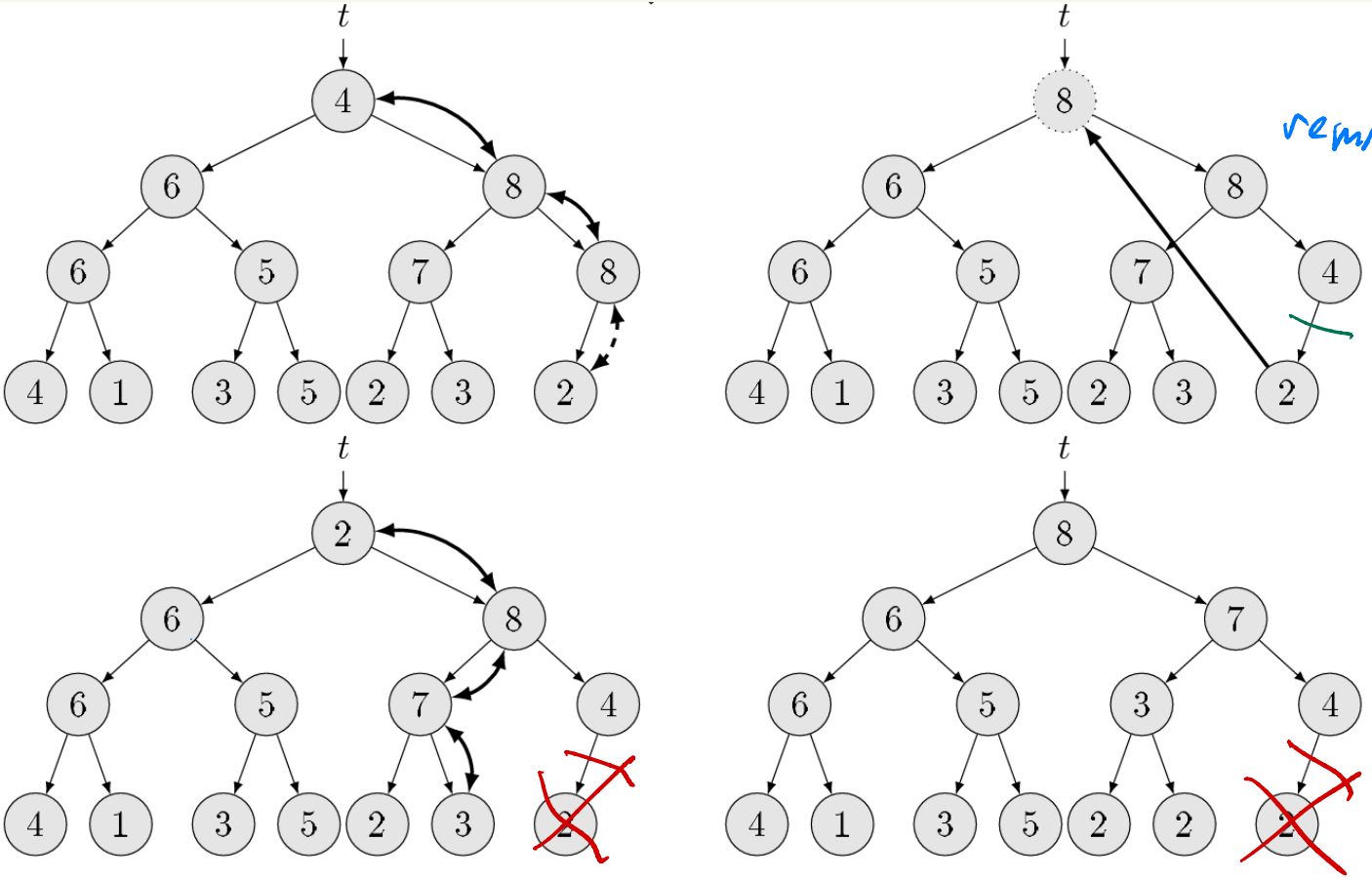


Figure 21: Heap operation visualization based on Figure 20:
add(8), add(2), add(9), remMax()=9, remMax()=8.

PrQueue::add($x : \mathcal{T}$)

$n = A.length$	
doubleFullArray(A)	SKIP
j:=n ; n:=n+1	
$A[j] := x$; $i := parent(j)$	
$j > 0 \wedge A[i] < x$	
swap($A[i], A[j]$)	
$j := i$; $i := parent(i)$	

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

$MT_{add:n=A.length}^{(n)} \in \Theta(n + \log n)$ $\quad \Theta(n)$ (= length n)

$AT_{add}^{(n)} \in \Theta(\log n)$

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

PrQueue::remMax() : $\mathcal{T}$

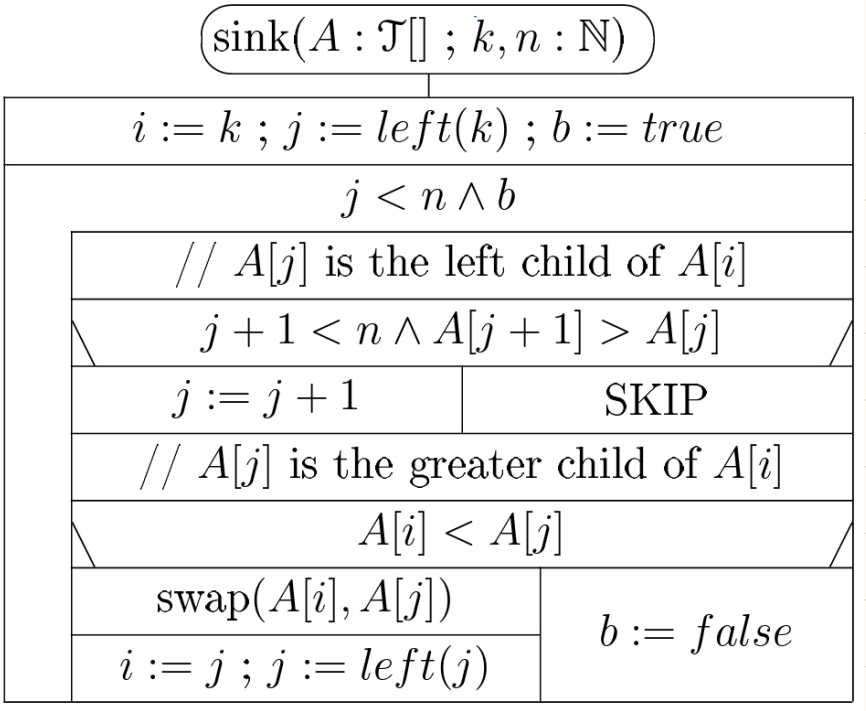
$n > 0$	
$max := A[0]$	PrQueueUnderflow
$n := n - 1$; $A[0] := A[n]$	
sink($A, 0, n$)	
return max	

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

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

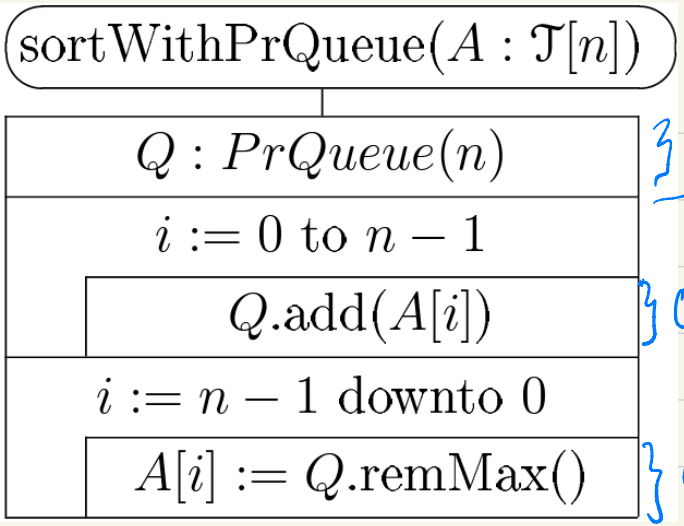
$+ \Theta(1)$

$= \Theta(\log n)$



$MT_{sink}^{(n)} \in \Theta(\log n)$

$mT_{sink}^{(n)} \in \Theta(1)$



$\Theta(1)$

$O(n \cdot \log n)$

$O(\log n)$

$O(n \cdot \log n)$

$O(\log n)$

$O(n \cdot \log n)$

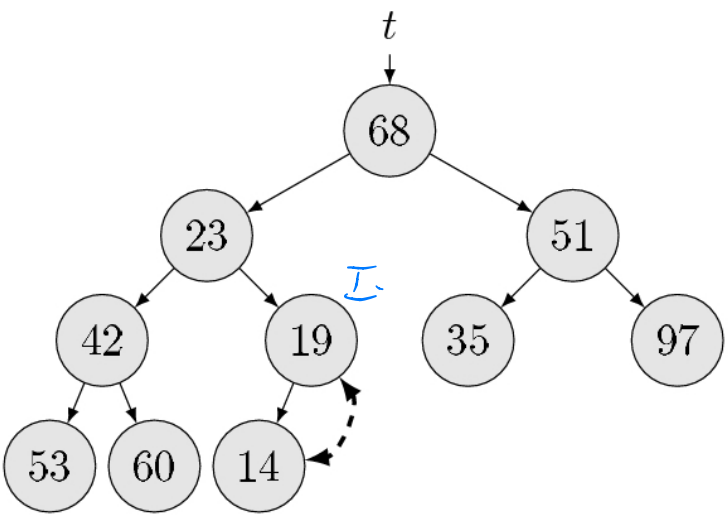
$MT(n) \in \Theta(n \cdot \log n)$

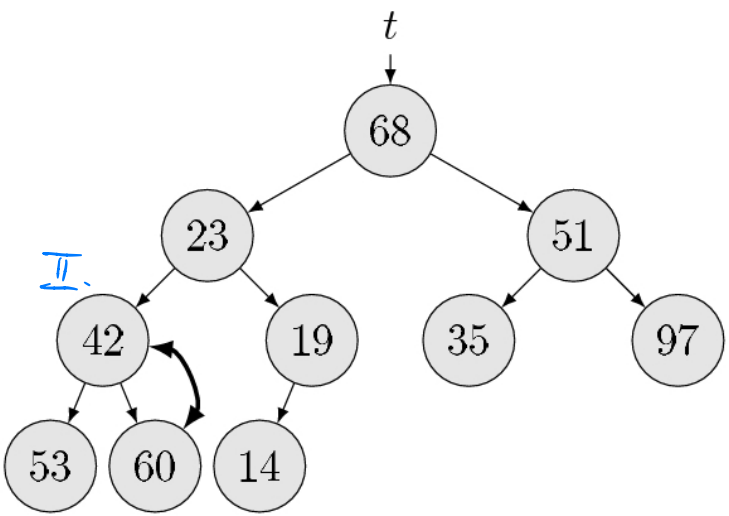
T, öh. rend

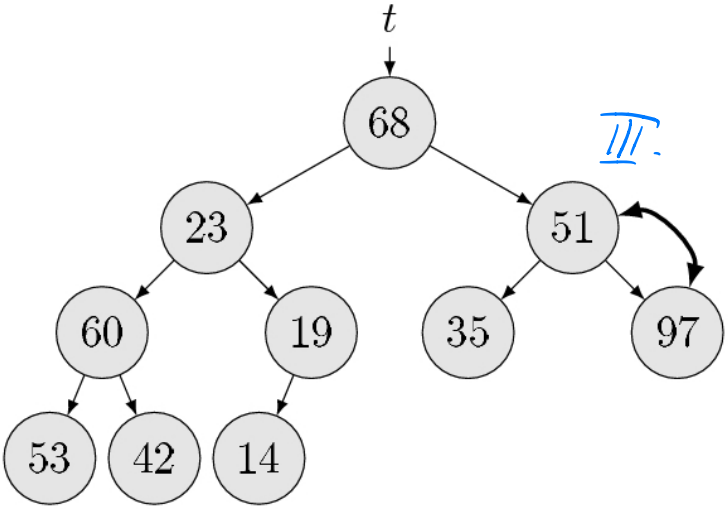
$S(n) \in \Theta(n)$

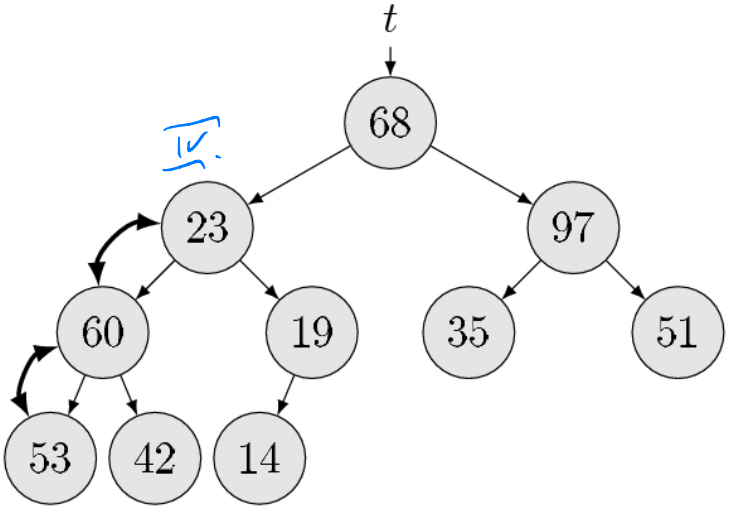
S öh. rend. $\Rightarrow MT_S(n) \in \Omega(n \cdot \log n)$

HEAPSORT (Kupacrendezés)


t
68
23
51
42
19
35
97
53
60
14
I.


t
68
23
51
II.
42
19
35
97
53
60
14


t
68
III.
23
51
60
19
35
97
53
42
14


t
68
IV.
23
97
60
19
35
51
53
42
14

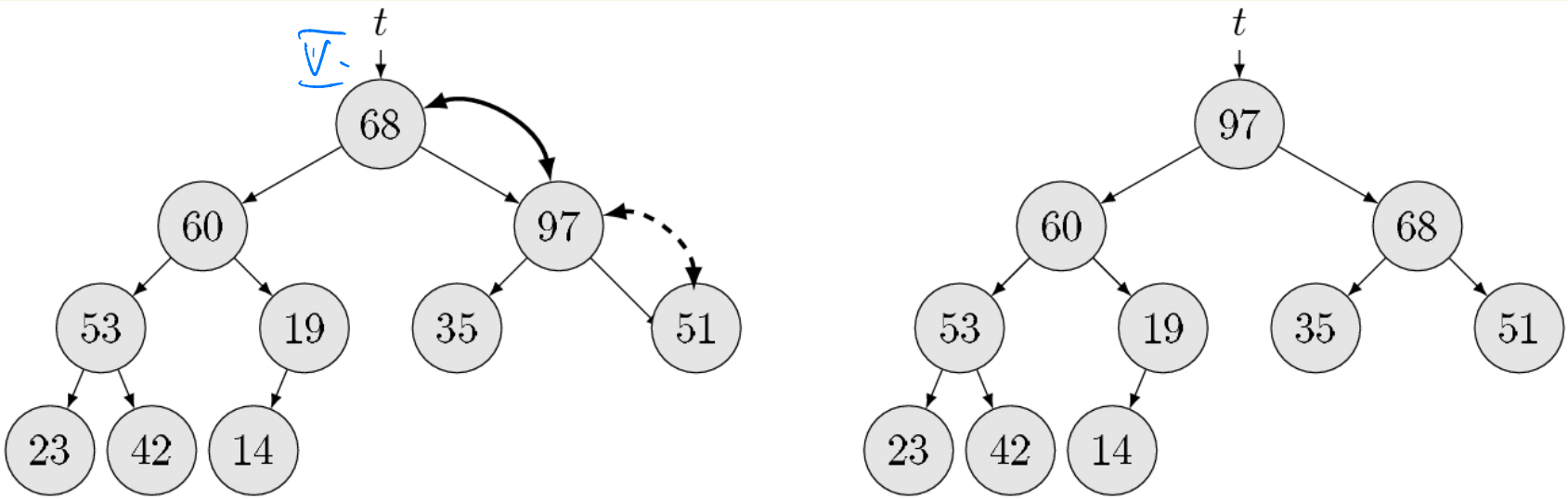


Figure 22: From array $\langle 68, 23, 51, 42, 19, 35, 97, 53, 60, 14\rangle$ we build a maximum heap. Notice that we work on the array from the beginning to the end. The binary trees show the logical structure of the array. Finally, we receive array $\langle 97, 60, 68, 53, 19, 35, 51, 23, 42, 14\rangle$. Dashed arrows compare where the sinking ends because no swap is needed.

$\approx \quad \frac{n}{4}$

$\approx \quad \frac{n}{8} \quad \frac{n}{8}$

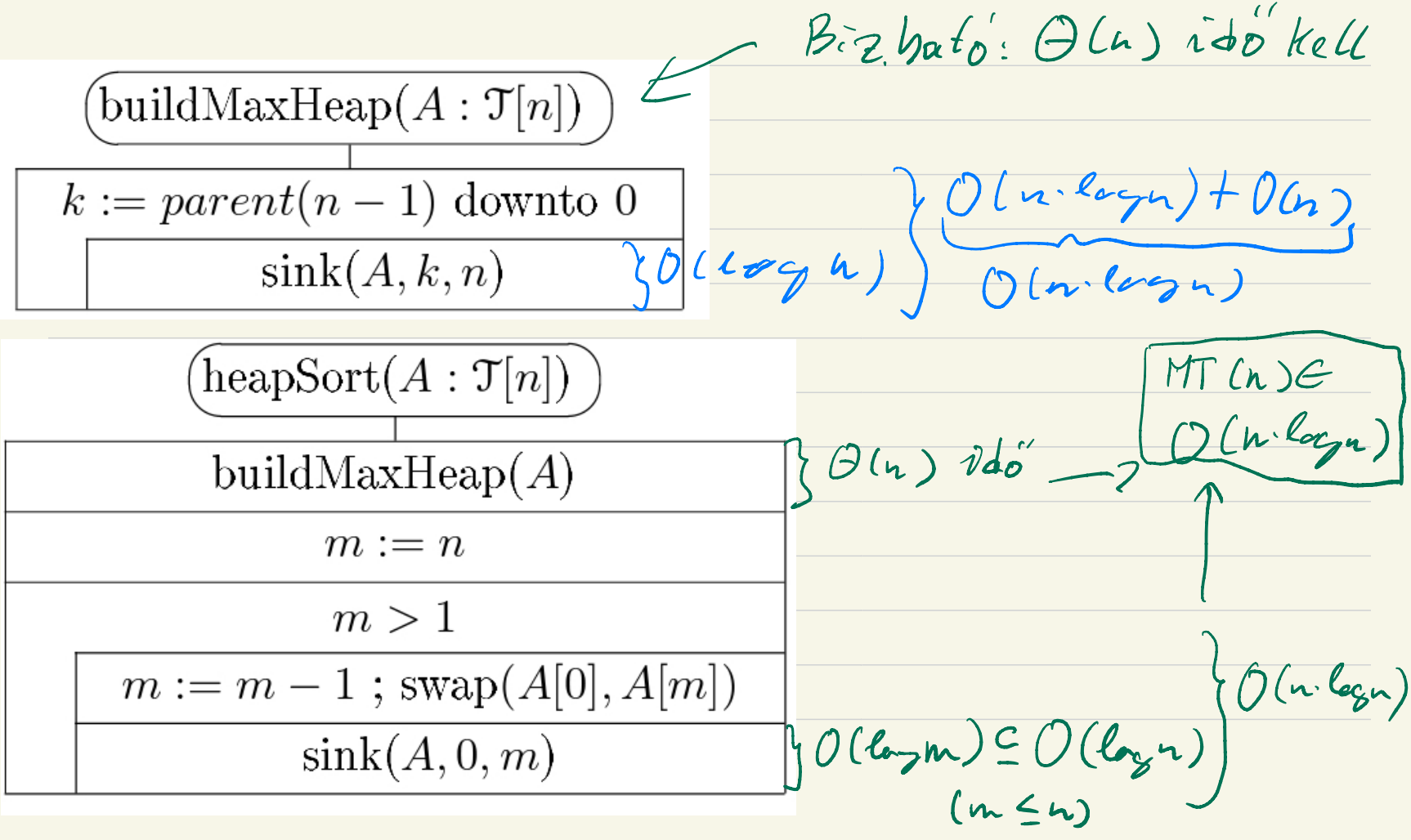
$\approx \quad \frac{n}{16} \quad \frac{n}{16} \quad \frac{n}{16}$

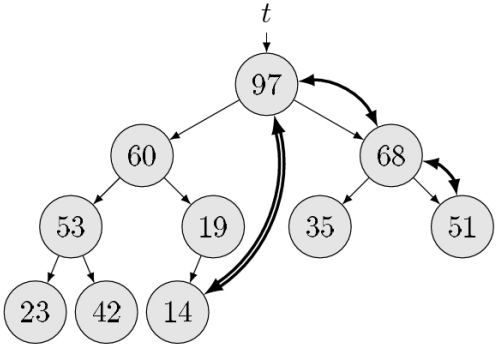
$\approx \quad \frac{n}{32} \quad \frac{n}{32} \quad \frac{n}{32} \quad \frac{n}{32}$

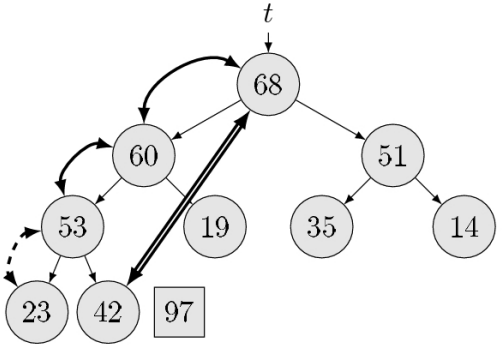
$\vdots \quad 1 \quad \vdots \quad \vdots$

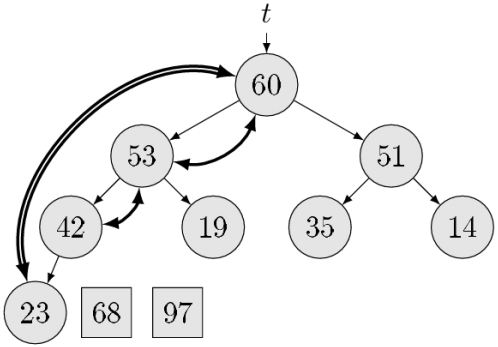
$$\frac{n}{2} + \frac{n}{4} + \frac{n}{8} + \frac{n}{16} + \dots = n \Big\} \text{ it}$$

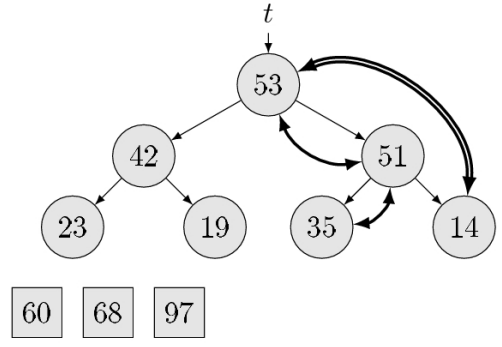
buildMaxHeap für uhl. $\lfloor \frac{n}{2} \rfloor$ $\Big\}$ $\Theta(n)$ mit. insgesamt

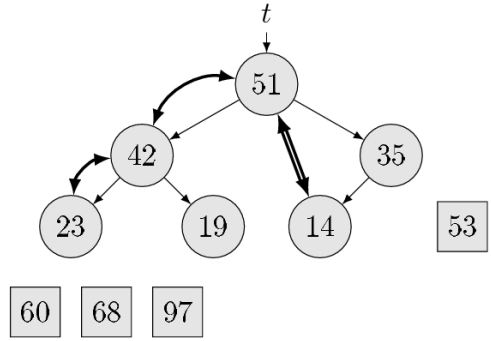

buildMaxHeap(A : T[n])
k := parent(n − 1) downto 0
sink(A, k, n)
Bizh.hato: Θ(n) idő kell
O(log n)
O(n · log n) + O(n)
O(n · log n)
heapSort(A : T[n])
buildMaxHeap(A)
m := n
m > 1
m := m − 1 ; swap(A[0], A[m])
sink(A, 0, m)
Θ(n) idő
MT(n) ∈ O(n · log n)
O(log m) ⊆ O(log n)
(m ≤ n)
O(n · log n)

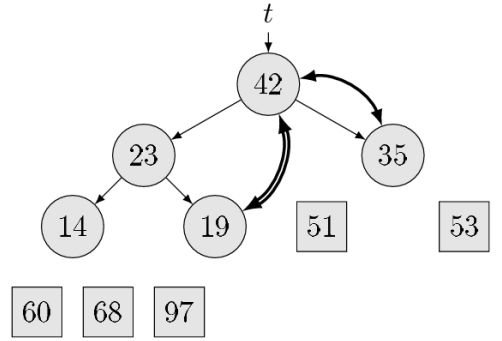

t
97
60
68
53
19
35
51
23
42
14


t
68
60
51
53
19
35
14
23
42
97


t
60
53
51
42
19
35
14
23
68
97


t
53
42
51
23
19
35
14
60
68
97


t
51
42
35
23
19
14
53
60
68
97


t
42
23
35
14
19
51
53
60
68
97

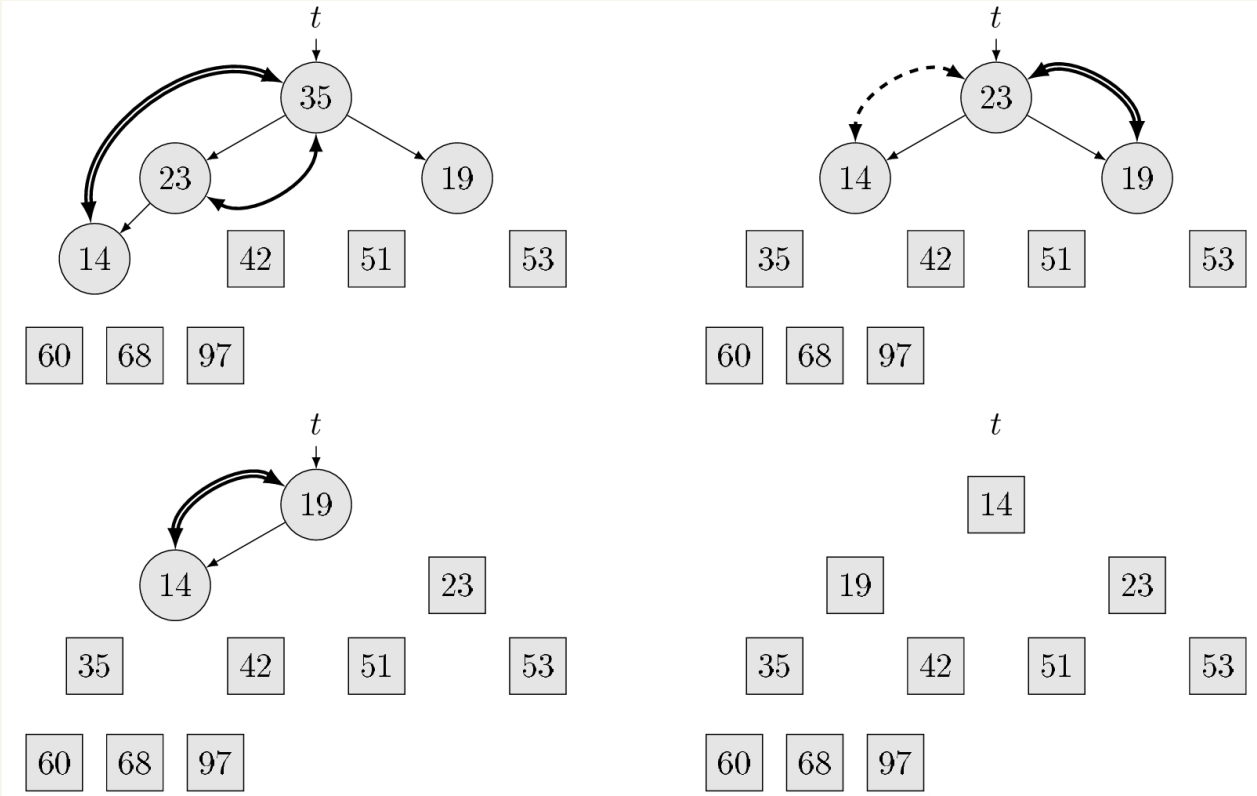


Figure 23: Heap sort visualization based on Figure 24. Double arrows represent the swap of the largest element with the last element of the heap, fitting into its desired place. The subsequent arrows demonstrate the sinking process after the initial swap. Dashed arrows compare where the sinking ends because no swap is needed.

$MT_{HS}(n) \in O(n \cdot \log n)$

$MT_{HS}(n) \in \Omega(n \cdot \log n)$ ↑

$\Big\} \Rightarrow MT_{HS}(n) \in \Theta(n \cdot \log n)$

$\forall$ összehasonlító rendezésre igaz

$\Downarrow$ (ÖH. rend.-ek alaptétele)

$MT_{HS}(n)$
$MT_{MS}(n)$
$MT_{QS+IS+HS}(n)$

$\Big\} \in \Theta(n \cdot \log n)$ aszimptotikusan optimális ÖH rendezések

(Az alaptétel szerint nem lehet jobb.)