

Programovacie techniky

4. stabilné triedenie, heap sort, ($n \log n$): vieme triediť rýchlejšie?

Porovnanie

PC: 10^8 porovnaní za sekundu

Superpočítač: 10^{12} porovnaní za sekundu

Pre $n = 10^6$ a 10^9

	Insertion sort (n^2)	Merge sort ($n \log n$)	Quick sort ($n \log n$)
PC	2.8 hod ($n=10^6$) 317 rokov ($n=10^9$)	1 sekunda 18 min	0.6 sek 12 min
Superpočítač	1 sekunda 1 týždeň	okamžite okamžite	okamžite okamžite

Dobrý algoritmus a jeho implementácia sú dôležitejšie ako vlastníctvo superpočítača!

Stabilné triedenie

New York	NY
Chicago	IL
Detroit	MI
Buffalo	NY
Milwaukee	WI
Champaign	IL

Stabilné triedenie



Chicago	IL
Champaign	IL
Detroit	MI
New York	NY
Buffalo	NY
Milwaukee	WI

Champaign	IL
Chicago	IL
Detroit	MI
New York	NY
Buffalo	NY
Milwaukee	WI

Chicago	IL
Champaign	IL
Detroit	MI
Buffalo	NY
New York	NY
Milwaukee	WI

Champaign	IL
Chicago	IL
Detroit	MI
Buffalo	NY
New York	NY
Milwaukee	WI

Stabilné triedenie = relatívne poradie rovnakých prvkov je zachované.

Stabilné triedenie

A je pole

i, j sú indexové premenné

$\pi(i), \pi(j)$: pozícia $A[i], A[j]$ po triedení

ak $i < j$ a $A[i] = A[j]$,

potom počas a po triedení platí:

$$\pi(i) < \pi(j)$$

Stabilné triedenie

Bubble/insertion sort: $O(n^2)$

Merge sort: $O(n \log n)$

Prečo? Zachovať relatívne poradie počas merge fázy je jednoduché.

Quicksort nie je stabilný, stabilné verzie existujú, ale nie sú praktické (vyžadujú pomocné pole atď.)

C++: triediaci algoritmus

C++03: $O(n \log n)$ v priemernom prípade

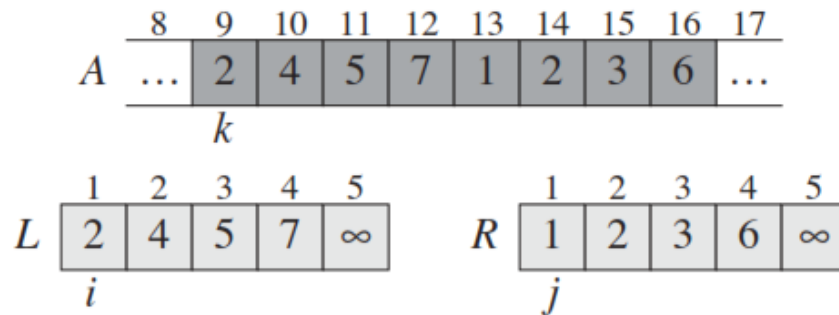
C++11: $O(n \log n)$ v najhoršom prípade

Štandard C++ nepredpisuje špecifický triediaci algoritmus

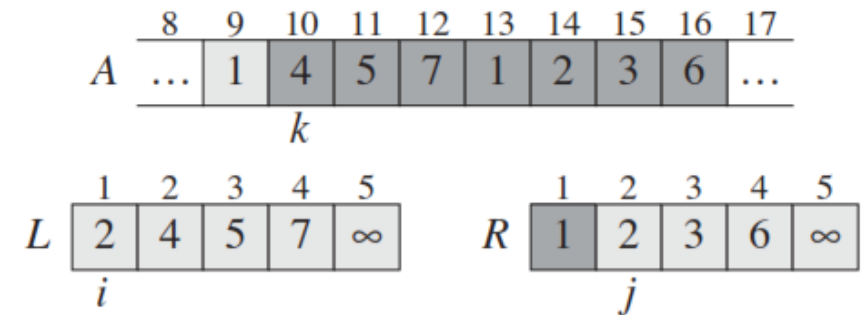
C++ obsahuje implementácie:

- `sort()`
- `stable_sort()` //iná časová zložitosť ako `sort()`

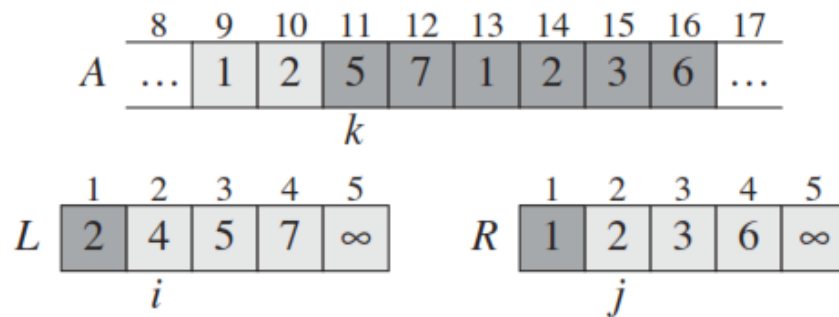
Merge sort: príklad merge



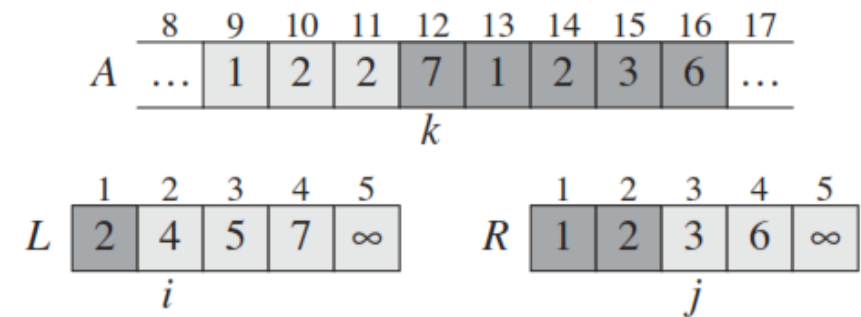
(a)



(b)

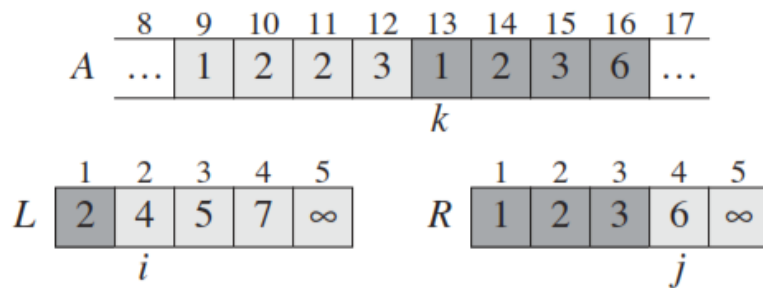


(c)

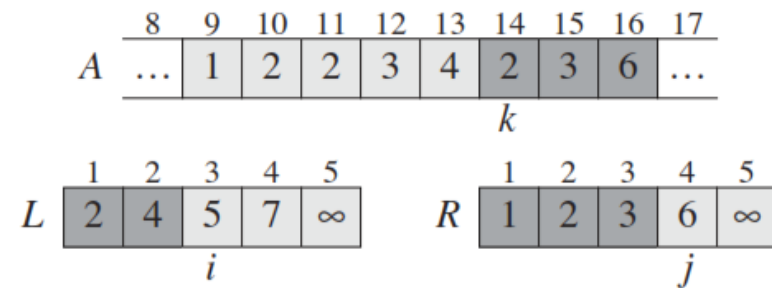


(d)

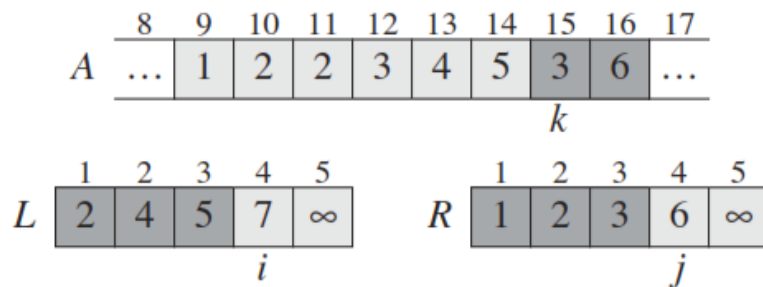
Merge sort: príklad merge



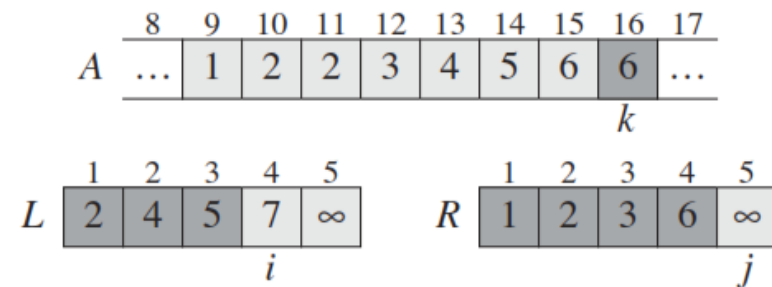
(e)



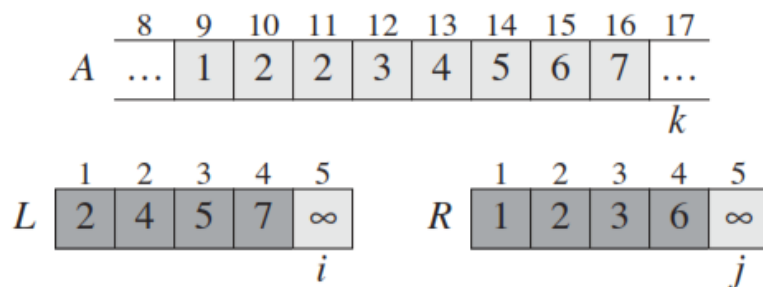
(f)



(g)



(h)



(i)

Merge: pseudokód

MERGE(A, p, q, r)

A je pole $A[p..q..r]$

```
1   $n_1 = q - p + 1$ 
2   $n_2 = r - q$ 
3  let  $L[1..n_1 + 1]$  and  $R[1..n_2 + 1]$  be new arrays
4  for  $i = 1$  to  $n_1$ 
5       $L[i] = A[p + i - 1]$ 
6  for  $j = 1$  to  $n_2$ 
7       $R[j] = A[q + j]$ 
8   $L[n_1 + 1] = \infty$ 
9   $R[n_2 + 1] = \infty$ 
10  $i = 1$ 
11  $j = 1$ 
12 for  $k = p$  to  $r$ 
13     if  $L[i] \leq R[j]$ 
14          $A[k] = L[i]$ 
15          $i = i + 1$ 
16     else  $A[k] = R[j]$ 
17          $j = j + 1$ 
```

Quick sort: partition

PARTITION(A, p, r)

```
1   $x = A[r]$ 
2   $i = p - 1$ 
3  for  $j = p$  to  $r - 1$ 
4      if  $A[j] \leq x$ 
5           $i = i + 1$ 
6          exchange  $A[i]$  with  $A[j]$ 
7  exchange  $A[i + 1]$  with  $A[r]$ 
8  return  $i + 1$ 
```

x je pivot

A je pole $A[p..r]$

Heap sort

Často používaný triediaci algoritmus

Časová zložitosť **vždy** $O(n \log n)$

Quick sort je v najhoršom prípade $O(n^2)$, v priemere $O(n \log n)$

Heap sort je lepší kandidát v časovo kritických prípadoch, ale Quick sort je väčšinou rýchlejší

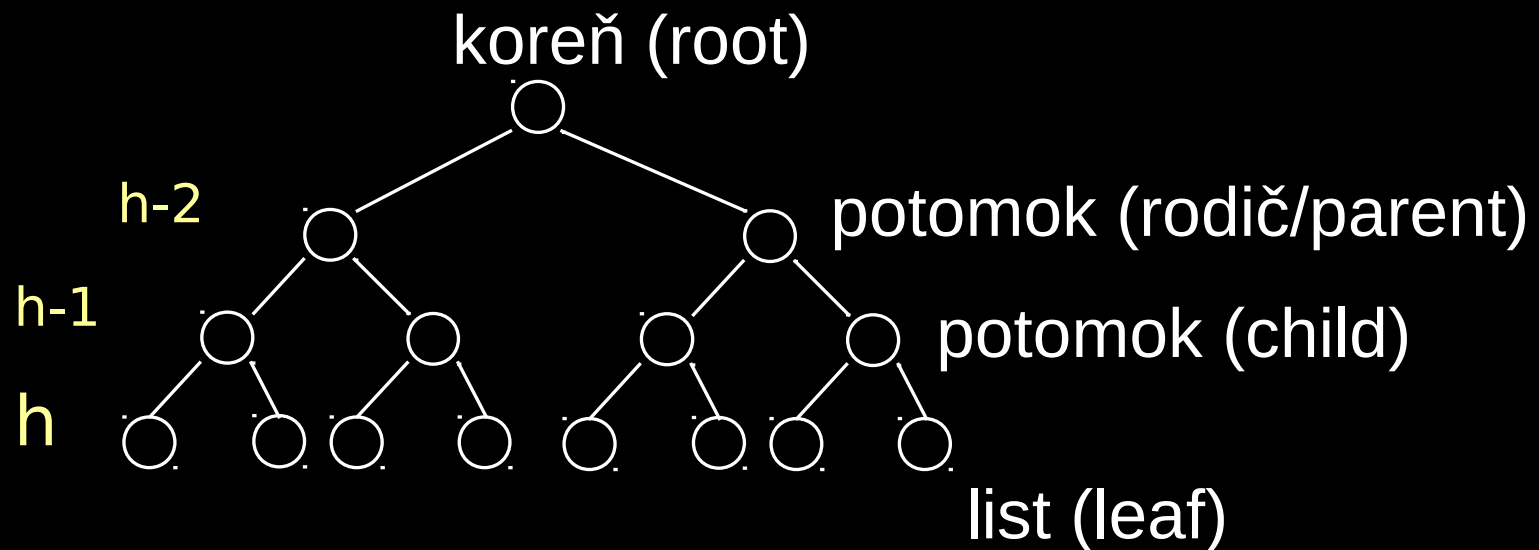
Heap sort je **cool**...

Heap sort: J. W. J. Williams



1964: foto je t'ážko nájst'

Binárny strom s hĺbkou h



Uzol: koreň, potomok, list...

Binárny strom: každý uzol, okrem listov, má dvoch potomkov

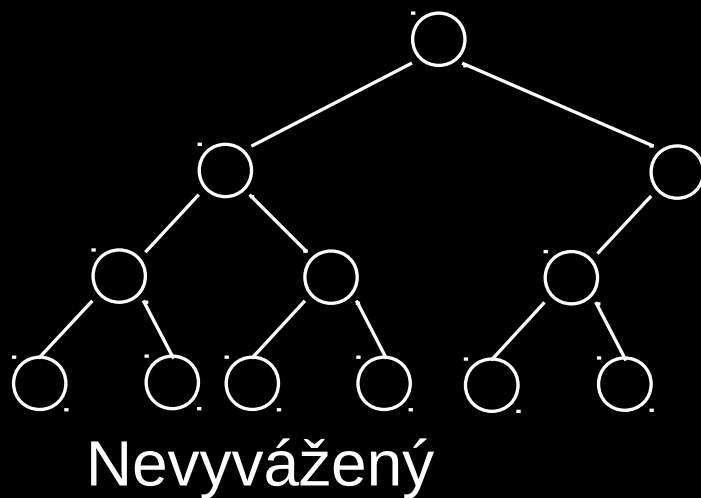
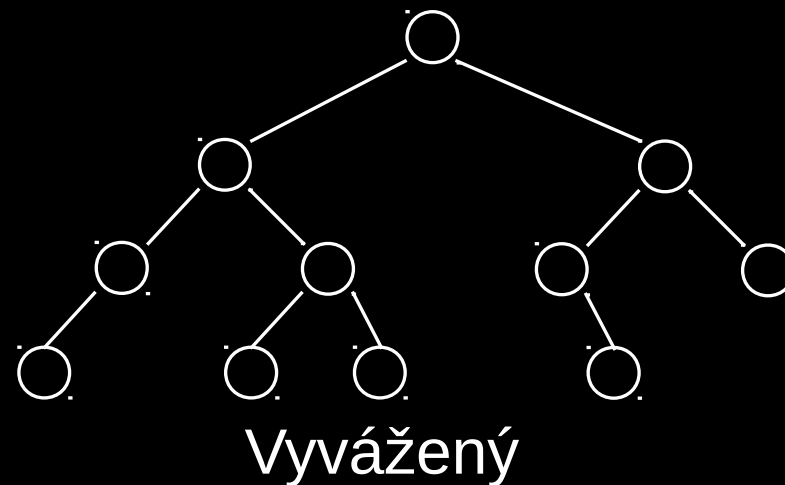
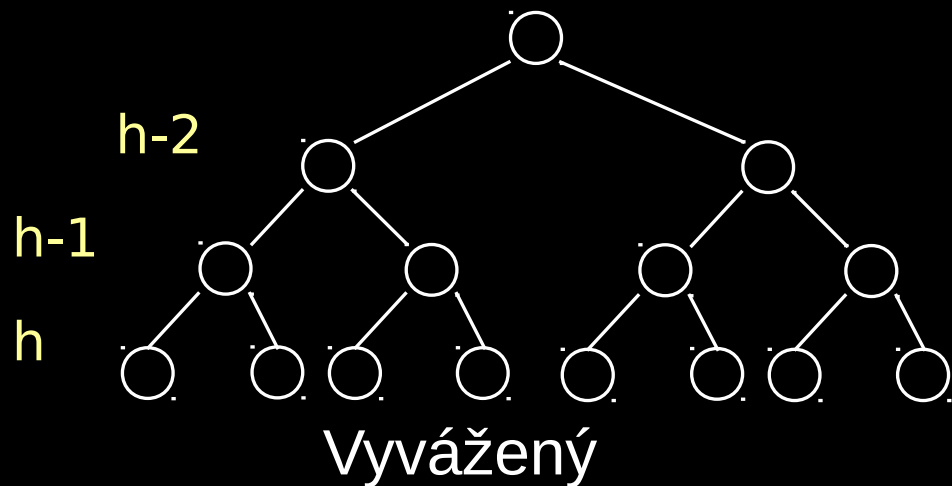
Vyvážený binárny strom

Hĺbka uzla: vzdialenosť od koreňa

Hĺbka stromu: vzdialenosť najvzdialenejšieho uzla od koreňa

Binárny strom s hĺbkou h je vyvážený, ak všetky jeho uzly s hĺbkou 0 až $h-2$ majú dvoch potomkov

Vyvážený binární strom

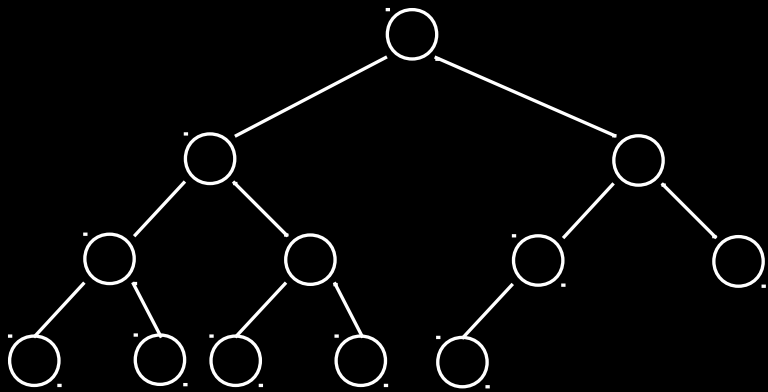


Dol'ava zarovnaný vyvážený..

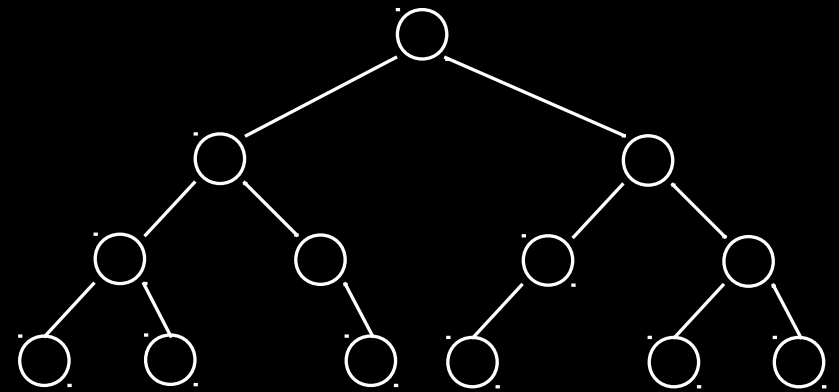
Vyvážený binárny strom je dol'ava zarovnaný ak:

- všetky listy majú rovnakú hĺbku, alebo
- všetky listy s hĺbkou h sú vľavo od listov s hĺbkou $h-1$

Vyvážený binárny strom



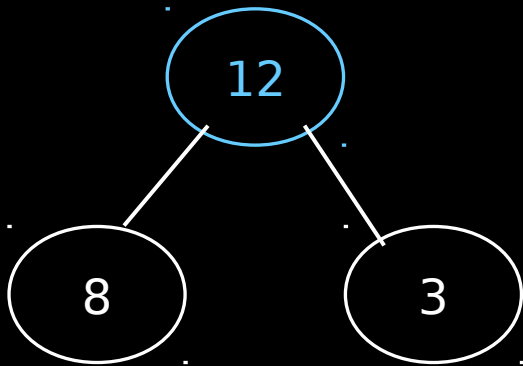
Doľava zarovnaný



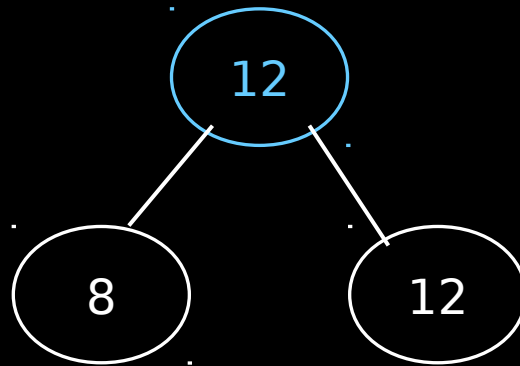
Nie je doľava zarovnaný

Heap vlastnosť

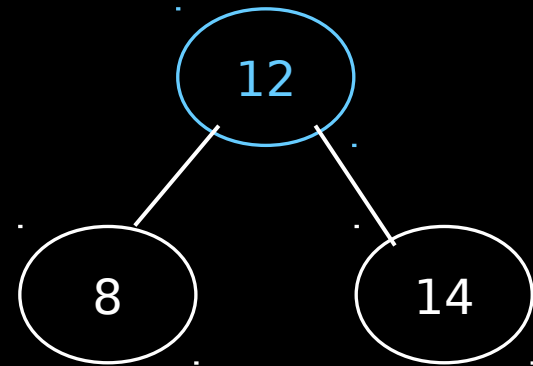
Hodnota v uzle je aspoň taká ako hodnota v potomkoch



Vlastnosť
splnená



Vlastnosť
splnená

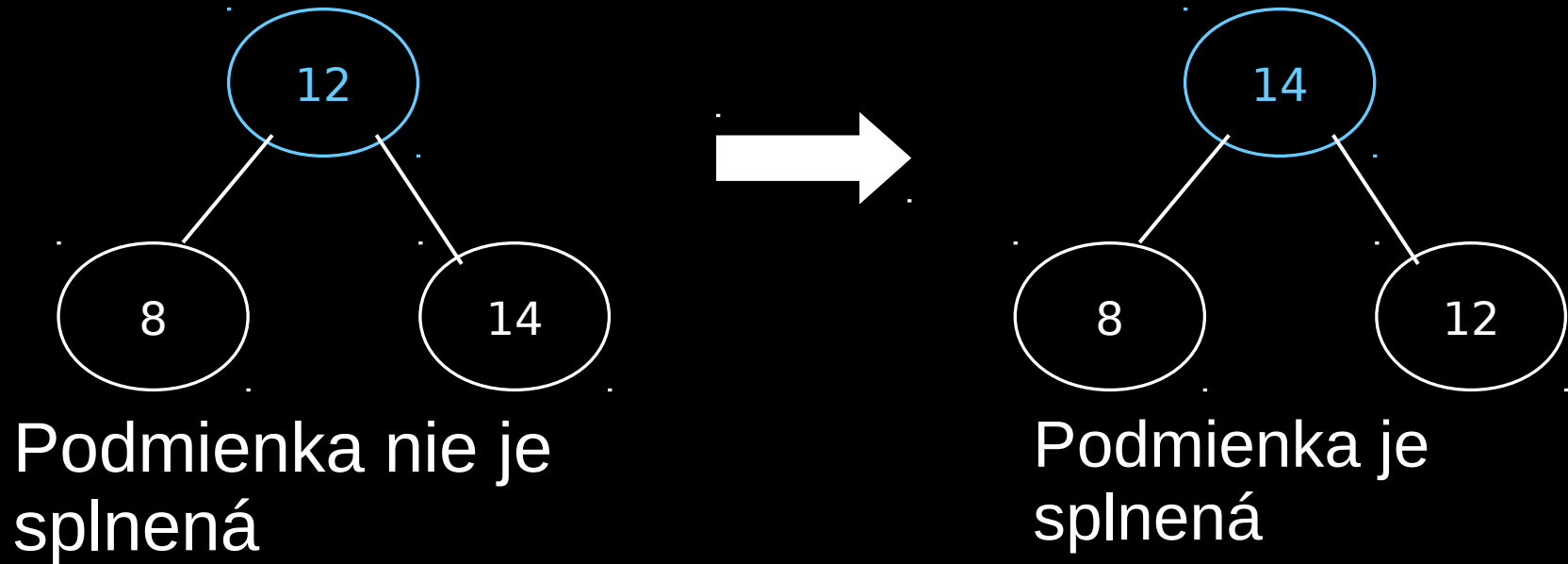


Vlastnosť nie je
splnená

Binárny strom je heap, ak je vlastnosť splnená pre všetky uzly

Heap = halda (asi čechizmus), kopa...

Porovnaj hodnotu uzla s hodnotou rodiča, vymeň ak má väčšiu hodnotu



Takáto výmena sa volá „sifting up“

Po výmene heap vlastnosť nemusí byť splnená

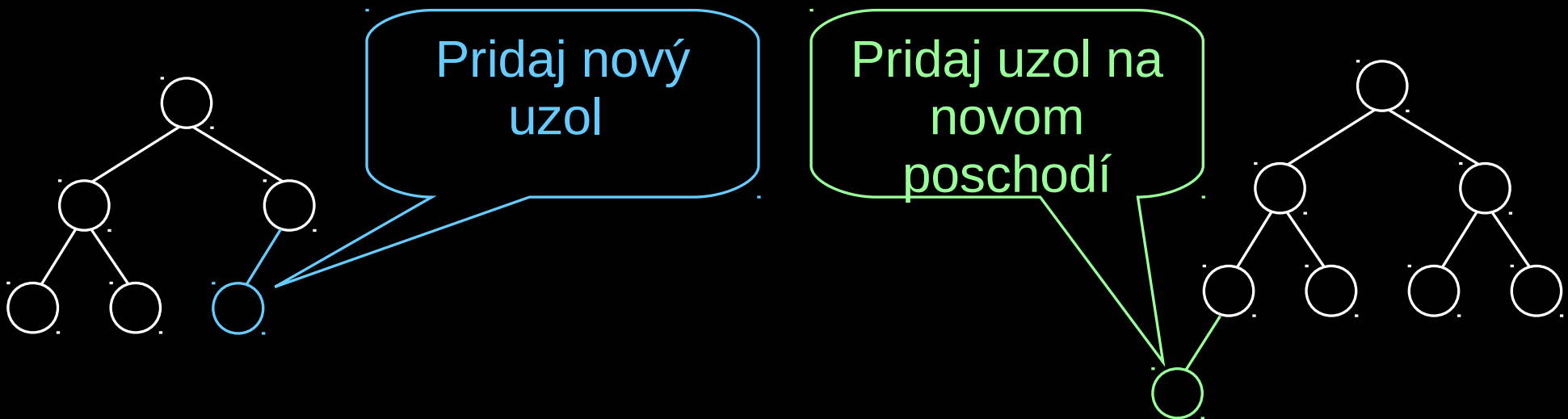
Heap: konštrukcia

Strom, ktorý má jeden uzol je heap

Ak treba pridať uzol:

Pridávaj uzol na najhlbšom poschodí na pravo
od uzlu čo je najviac vpravo

Ak na poschodí nie je voľné miesto, začni na
novom hlbšom poschodí



Heap: konštrukcia pokr.

Vždy keď pridáme uzol môžeme porušiť heap podmienku

Vždy keď pridáme uzol a porušíme heap vlastnosť musíme urobiť SiftUp

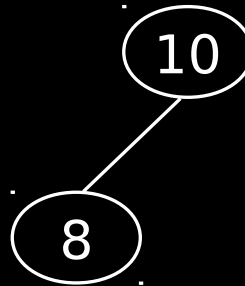
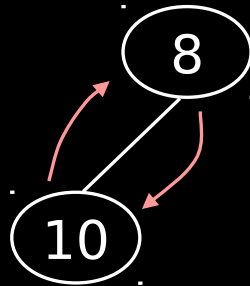
Ak urobíme SiftUp môžeme porušiť heap vlastnosť pre rodiča

SiftUp opakujeme až pokiaľ už nie je potrebný, alebo ak dosiahneme koreň

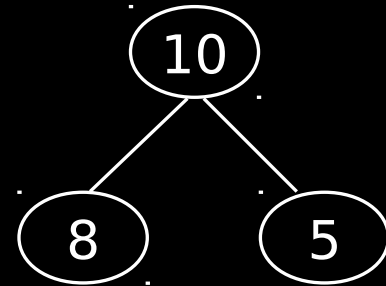
Heap: konštrukcia príklad



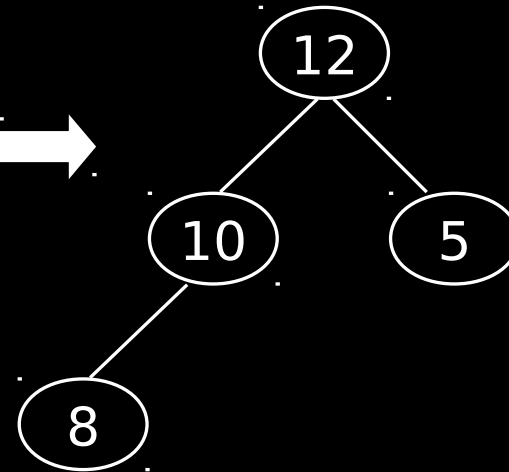
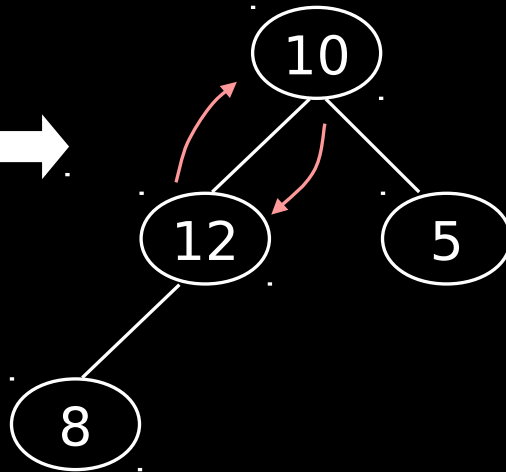
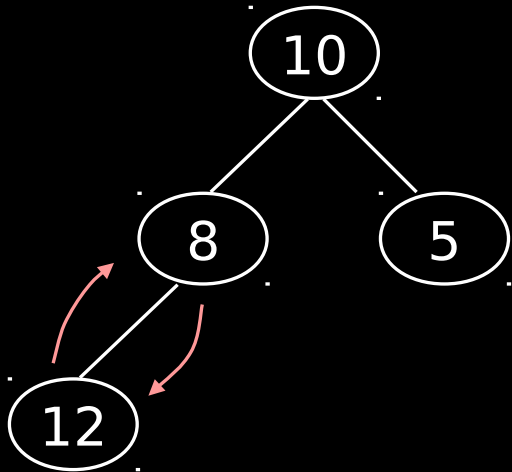
1



2

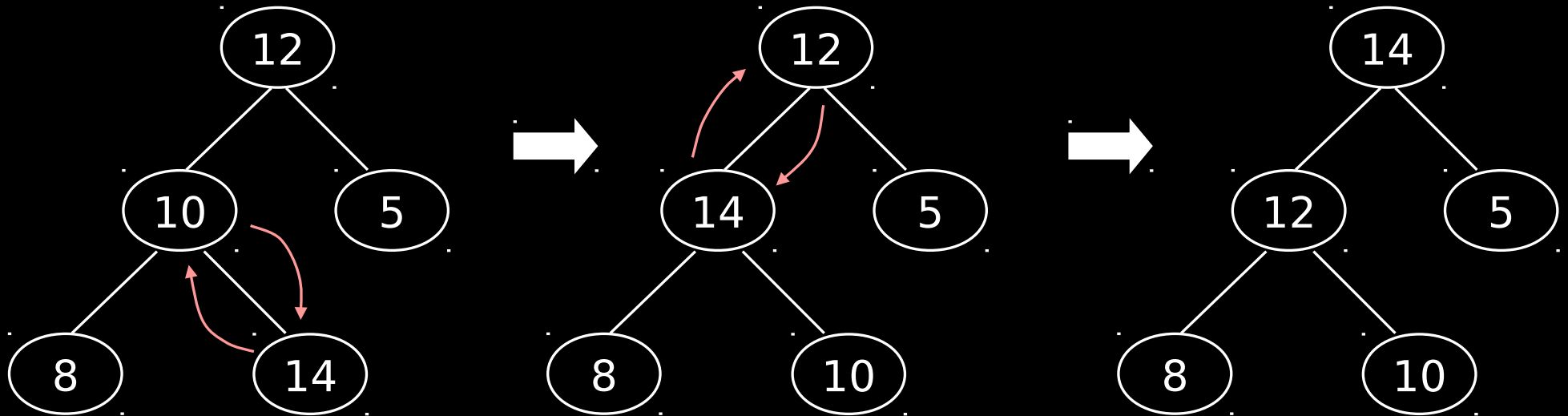


3

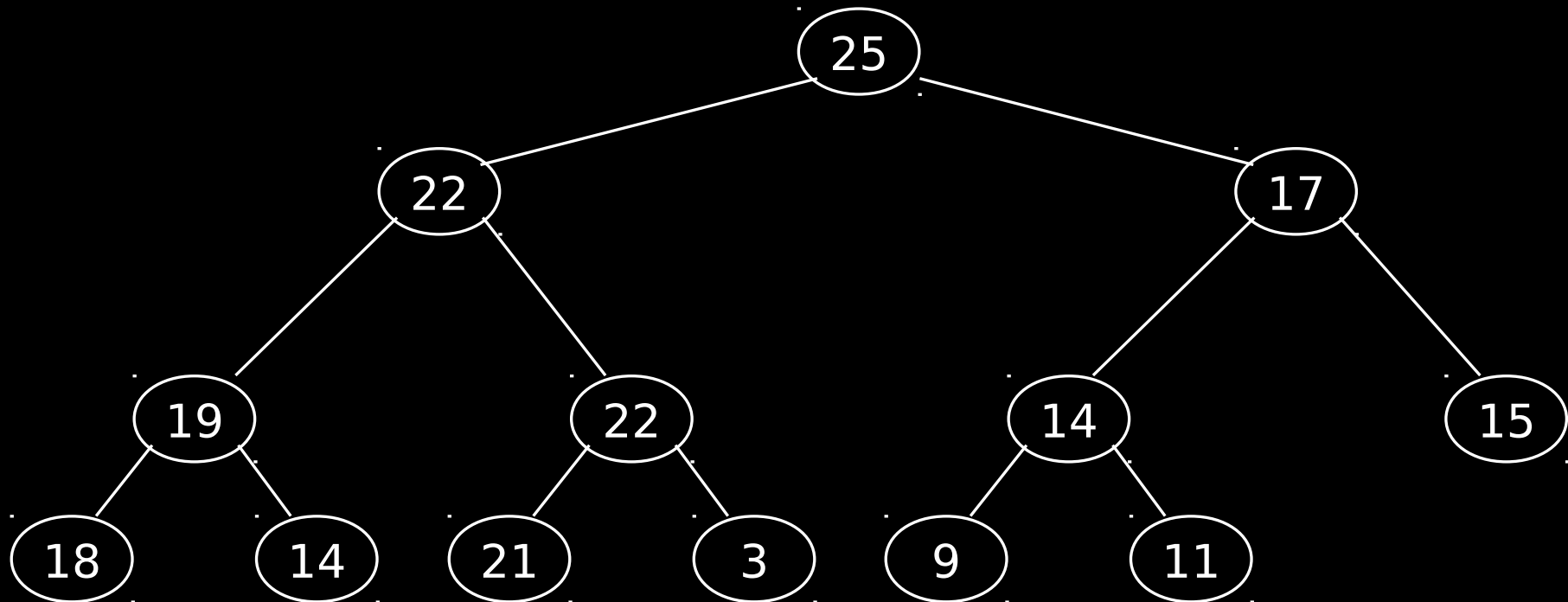


4

Heap: konštrukcia príklad



Heap: konštrukcia príklad

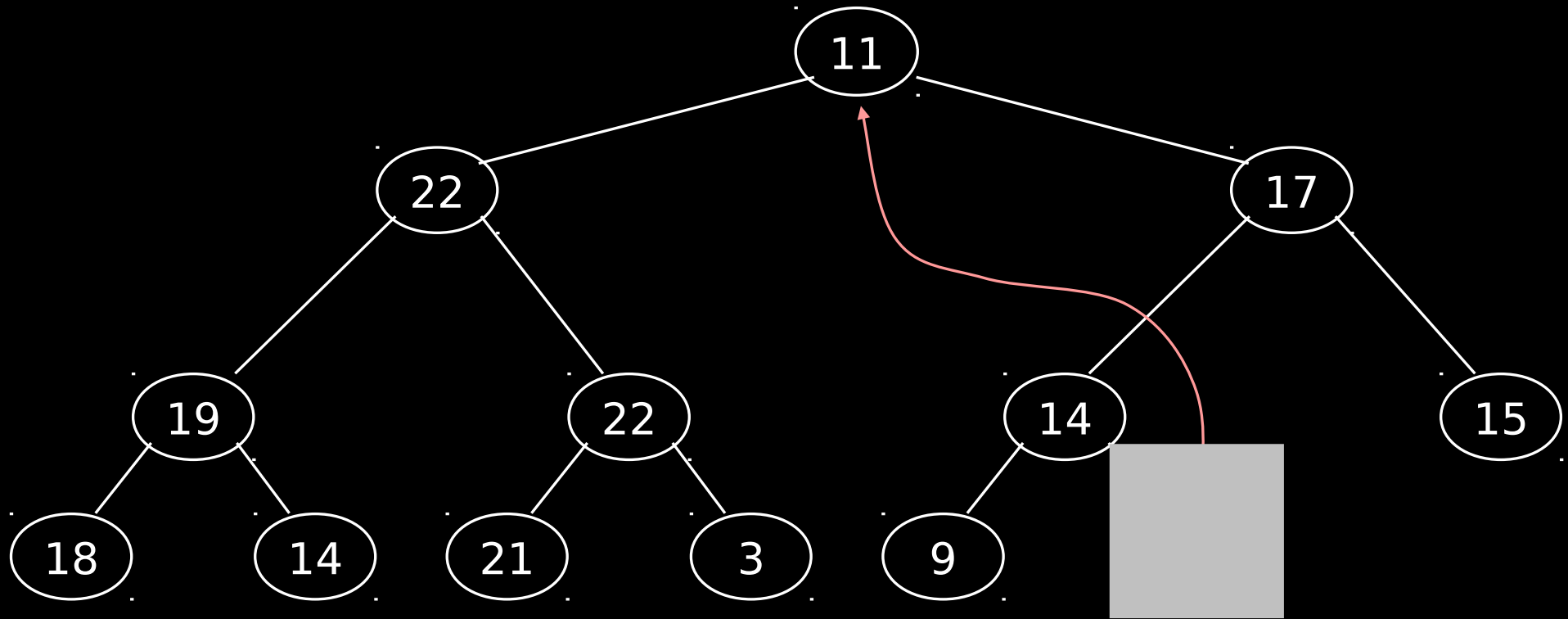


Binárny strom, ktorý spĺňa heap vlastnosť

Heap nie je roztriedený

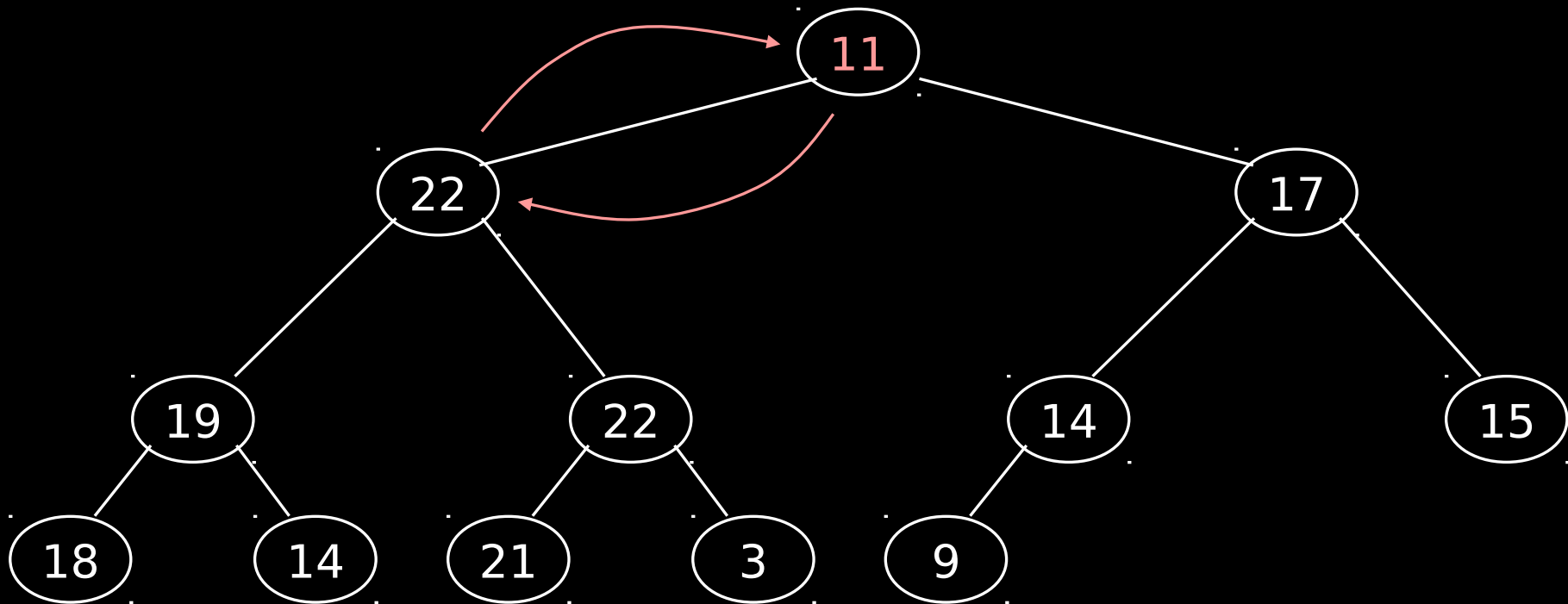
SiftUp nemení vzhľad binárneho stromu

Heap: odstránenie koreňa



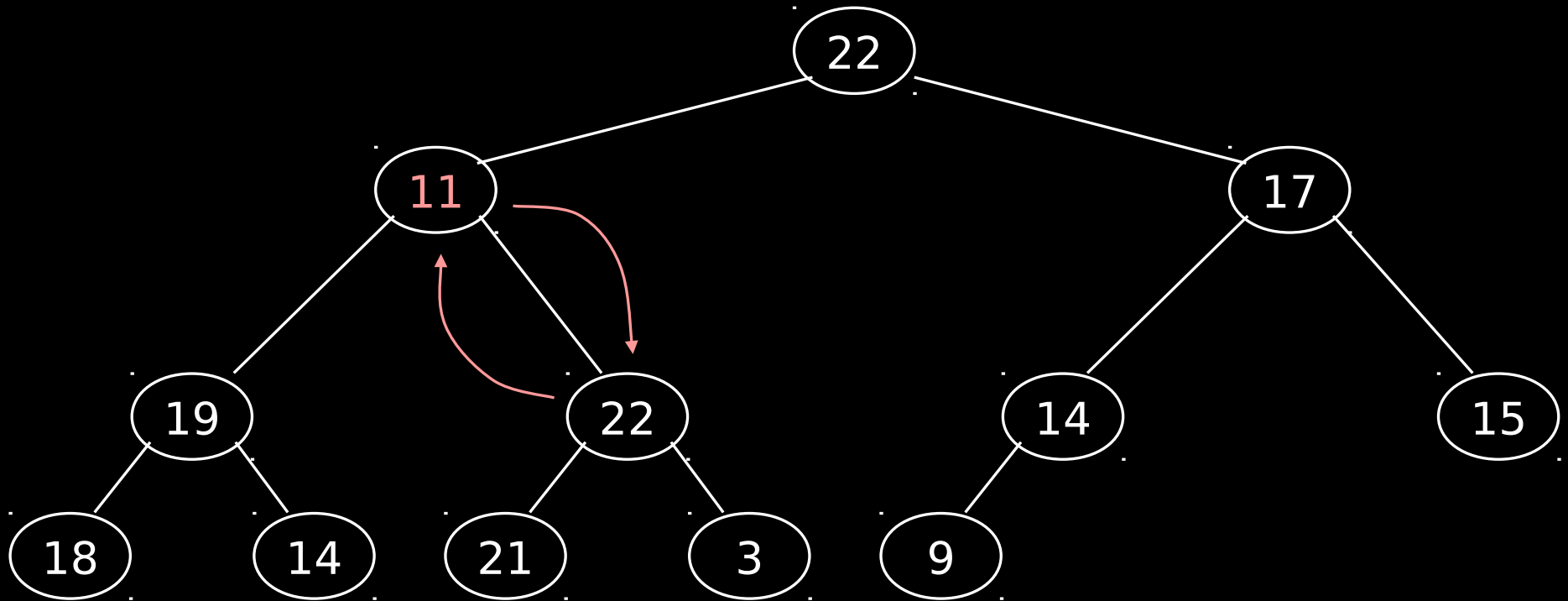
Čo musíme spraviť aby bol binárny strom
vyvážený a doľava zarovnaný? : list najviac
vpravo je nový koreň

reHeap

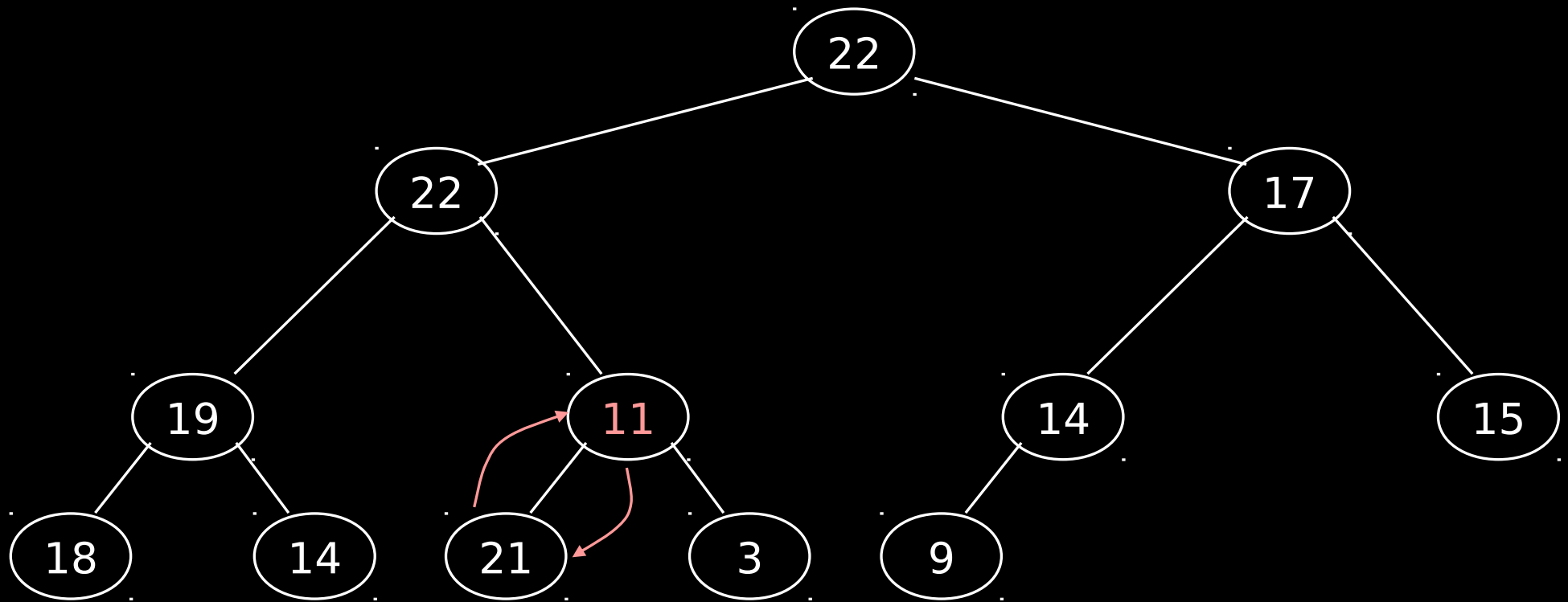


Koreň stratil heap vlastnosť: porovnaj s potomkami. Ak je heap podmienka porušená, potom vymeň s potomkom, ktorý je väčší.

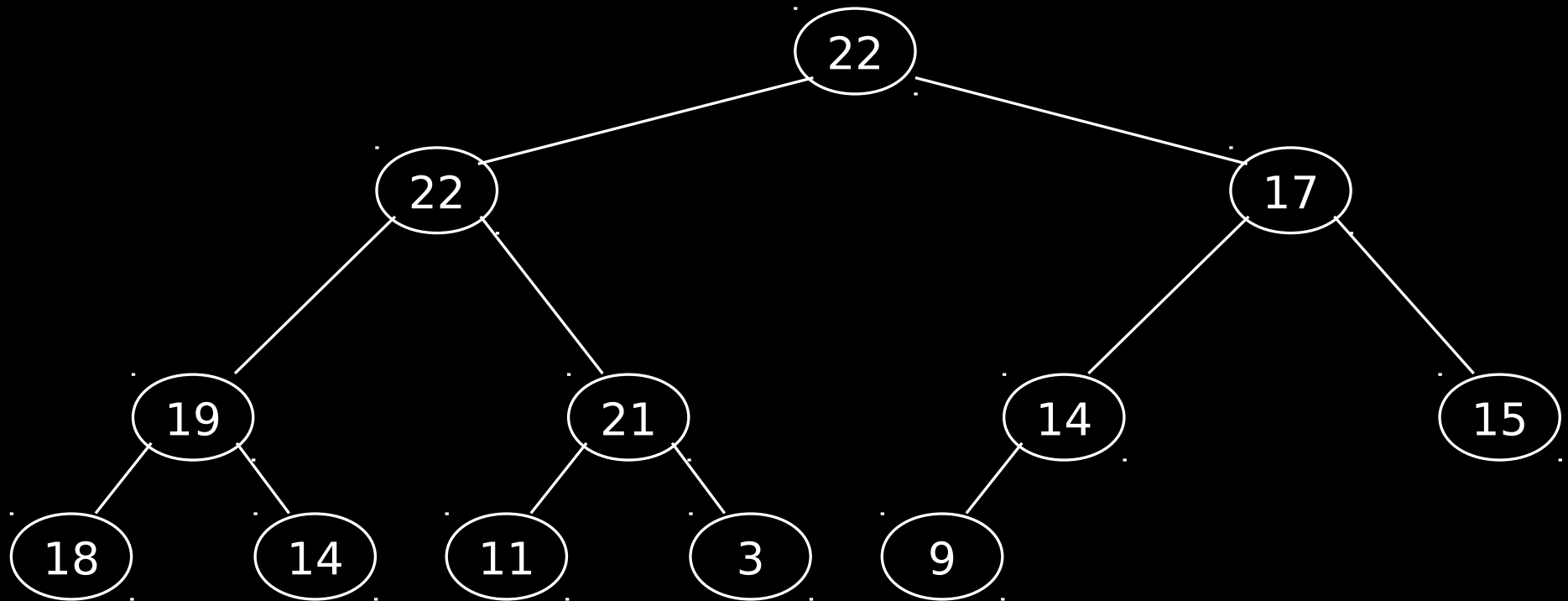
reHeap



reHeap

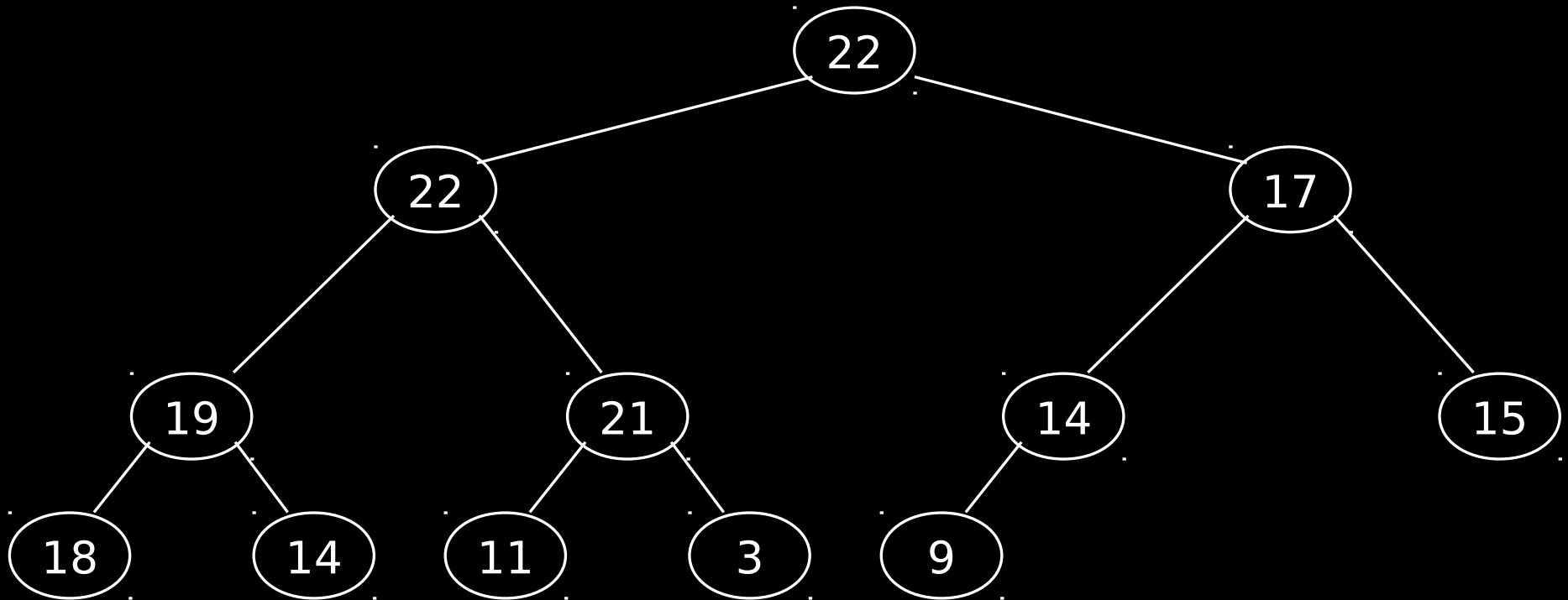


Heap



Koreň má najväčšiu hodnotu!!
22...(aha...)

Heap



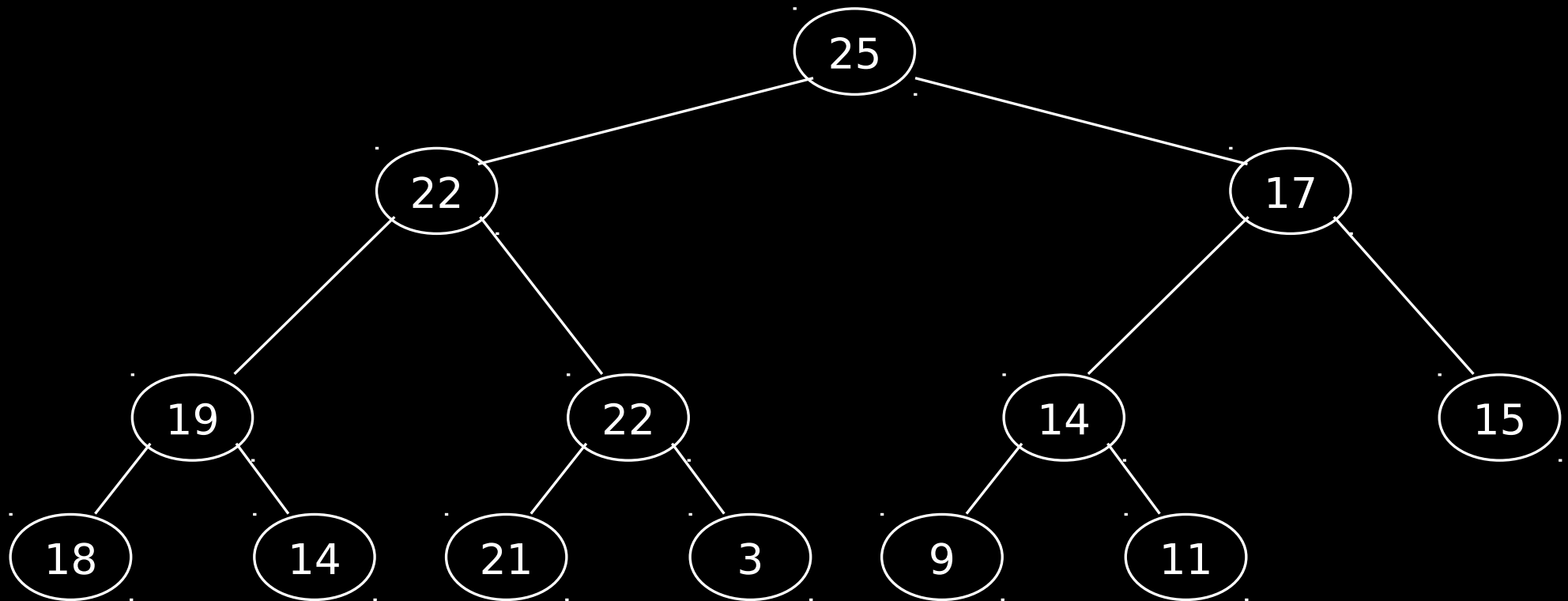
Odstraňuj koreň až pokým
binárny strom nie je prázdny!

Heap: triedenie

urob z poľa heap;

```
opakuj {  
    odstráň a nahrad' koreň;  
    reHeap;  
}
```

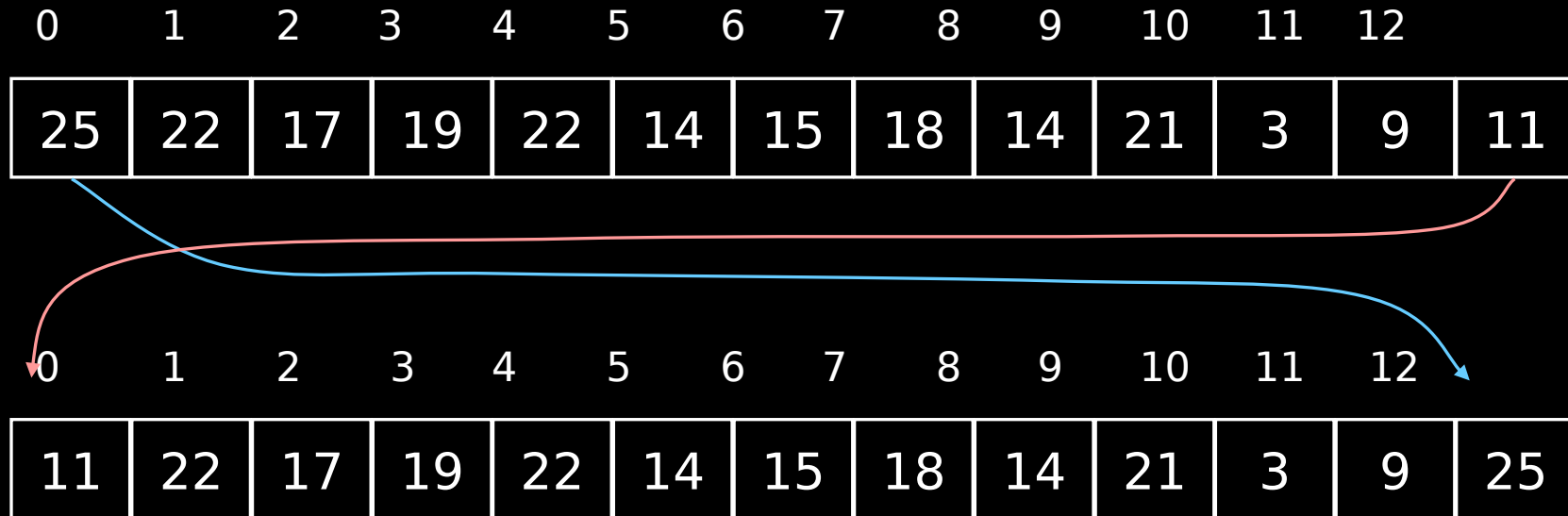
Pole ako heap



0 1 2 3 4 5 6 7 8 9 10 11 12

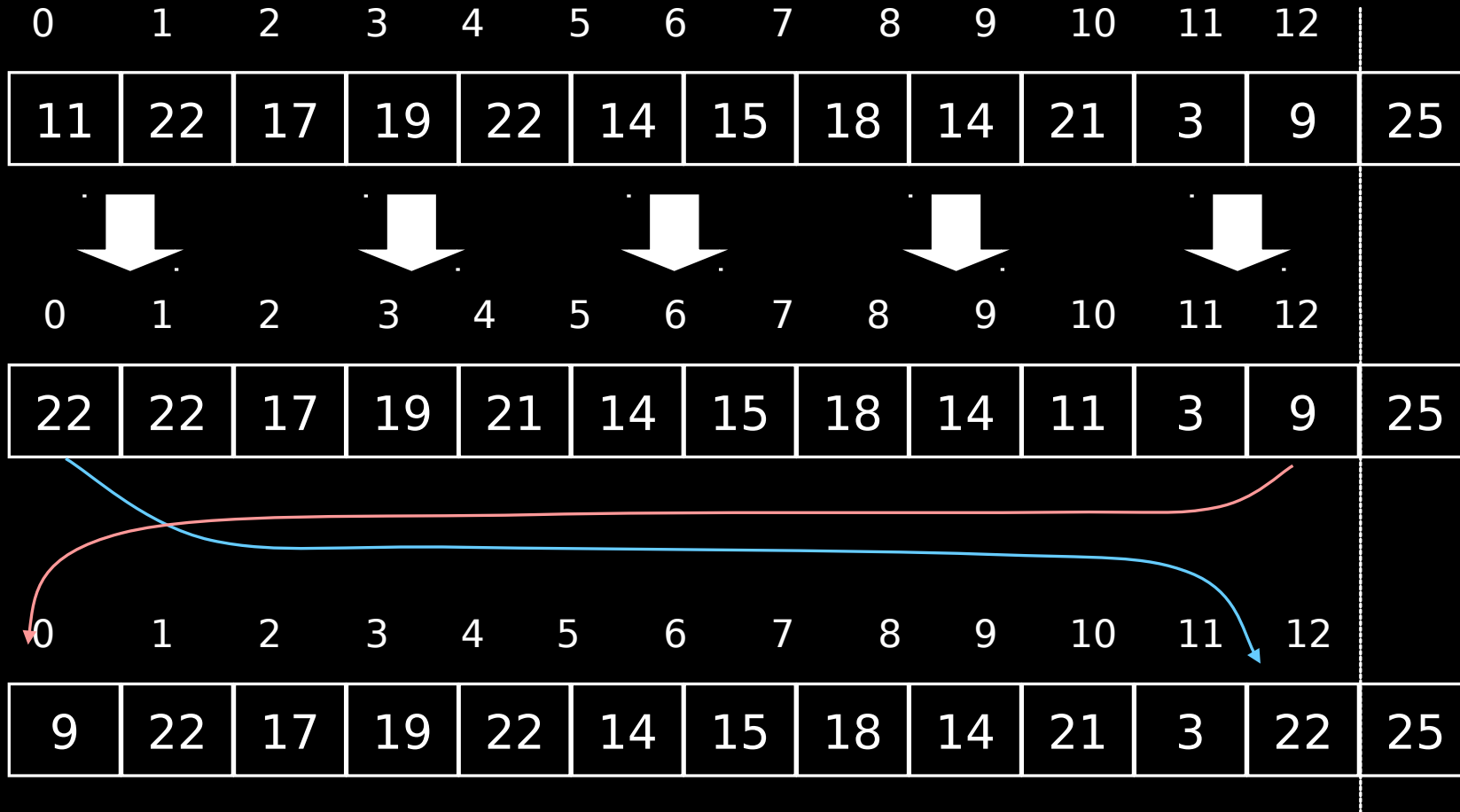
25	22	17	19	22	14	15	18	14	21	3	9	11
----	----	----	----	----	----	----	----	----	----	---	---	----

Ostráň a nahrad' koreň



Koreň je element, ktorý je najviac vľavo
Vymeň ho s elementom najviac vpravo
reHeap...

Pop reHeap



Heap: časová zložitosť

urob z poľa heap;

Opakujeme najviac n -krát (pre každý element)

SiftUp od lista ku koreňu: $\log n$

Binárny strom je vyvážený, jeho hĺbka je $(\log n)$

$O(n \log n)$

Heap: časová zložitost'

```
opakuj {  
    odstráň a nahrad' koreň;  $O(1)$   
    reHeap;  $O(\log n)$   
}
```

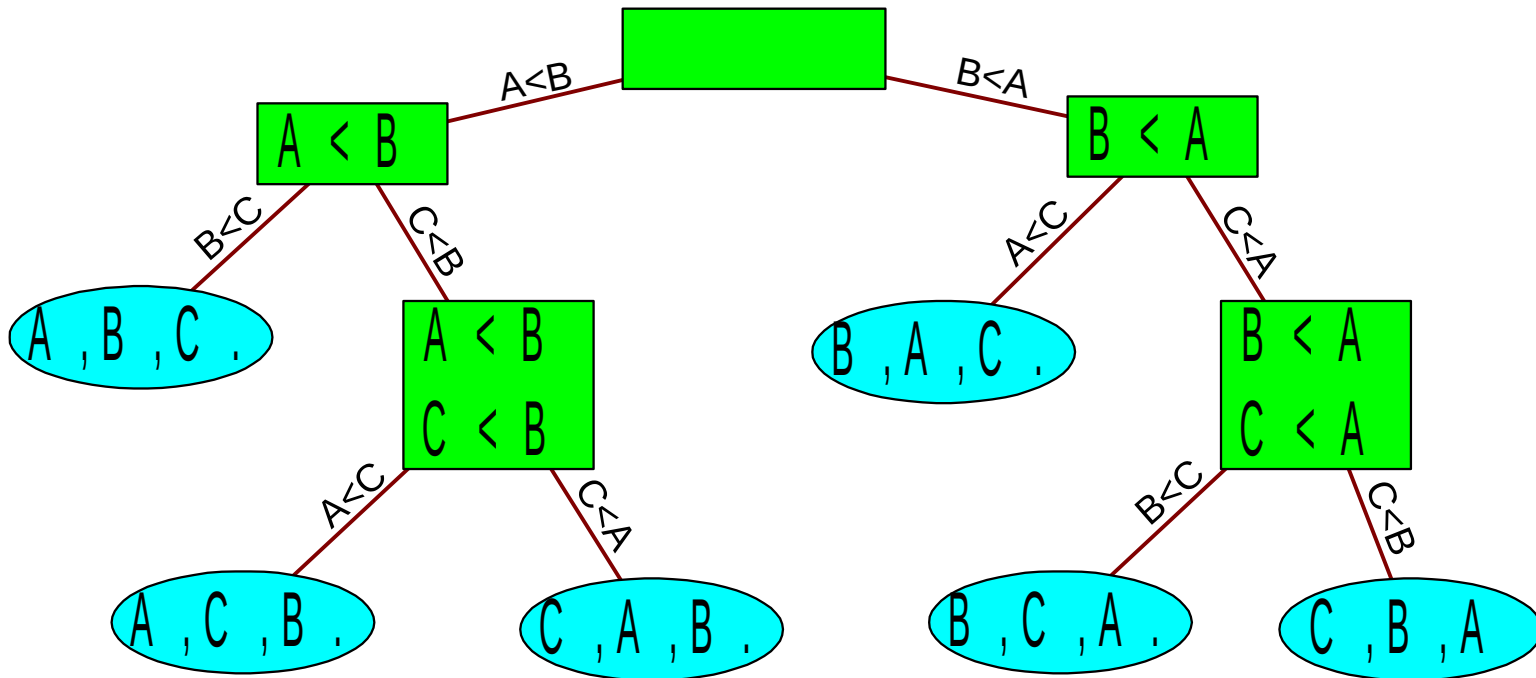
$O(n \log n)$

Priority queue

Heap implementuje abstraktný dátový
typ: priority queue

Operácia: **nájdi max (min) element**

Triedenie s porovnávaním



Koľko permutácií pre n elementov: $n!$

Akú hĺbku má binárny strom, ktorý má aspoň $n!$ listov: $\log n!$

Triedenie s porovnávaním

Stirlingova aproximácia: $n! \underset{+\infty}{\sim} \left(\frac{n}{e}\right)^n \sqrt{2\pi n}$

$$\log(n!) \approx n \log n$$

Stirlingova aproximácia nepresná pre malé n