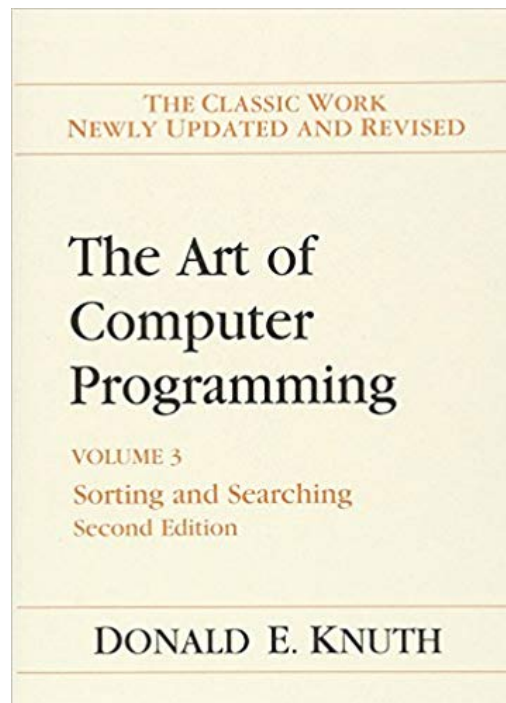


Algoritmizace

Třídění



Osnova

Třídění v poli

- přímé metody (BubbleSort, SelectionSort, InsertionSort)
- haldové třídění (HeapSort)

Složitost problému vnitřního třídění

Třídění v lineárním čase

Vnitřní třídění

Vstup: pole **a** s prvky, které lze porovnávat

Výstup: pole **a** s prvky uspořádanými vzestupně

Bublínkové třídění **BubbleSort**

- projdi pole a porovnej dvojice sousedních prvků
- v případě potřeby dvojici vyměň
- po dosažení konce seznamu začni znovu od začátku
- pokračuj až na pozici poslední výměny v předchozím kroku
- není-li již žádná dvojice pro výměnu, výpočet končí

Bublínkové třídění BubbleSort

```
def bubbleSort(a):  
    n = len(a)  
    while n > 1:  
        vymena = 0  
        for i in range(n-1):  
            if a[i] > a[i+1]:  
                a[i+1], a[i] = a[i], a[i+1]  
                vymena = i+1  
        n = vymena  
    return a
```

✗ Časová složitost $\Theta(n^2)$

Třídění výběrem SelectionSort

První krok

- najdi minimální prvek
- a vyměň s prvkem na pozici 0

Další krok

- mezi zbývajícím prvkem najdi minimální
- a vyměň s prvkem na pozici 1

Invariant cyklu

- po provedení i -tého kroku
- tvoří $a[0], a[1], \dots, a[i-1]$ setříděný úsek
- který obsahuje i minimálních prvků pole a

Třídění výběrem SelectionSort

```
def selectionSort(a):  
    for i in range(len(a) - 1):  
        # a[i] vyměň s minimem z a[j], j ≥ i  
        minIndex = i  
        for j in range(i+1, len(a)):  
            if a[minIndex] > a[j]:  
                minIndex = j  
        a[i], a[minIndex] = a[minIndex], a[i]  
    return a
```

Třídění výběrem – analýza

✖ **Proti:** Časová složitost $\Theta(n^2)$

Data malého rozsahu (desítky prvků)

- lepší nežli BubbleSort

✓ **Pro:**

- Jen $n - 1$ výměn
- jen $O(n)$ zápisů do pole **a**

Třídění vkládáním InsertionSort

Jako třídíme karty

- vezměte novou kartu z balíčku
- a postupným porovnáváním zprava doleva
- s již setříděnými kartami, které držíte v ruce
- vložte na správné místo



Třídění vkládáním InsertionSort

```
def insertionSort(a):  
    for i in range(1, len(a)):  
        # vlož a[i] do setříděného  
        # a[0..i-1]  
        x, j = a[i], i  
        while j > 0 and a[j-1] > x:  
            a[j] = a[j-1]  
            j -= 1  
        a[j] = x  
    return a
```

Invariant: po i -tém kroku je $a[0..i]$ setříděno

Třídění vkládáním – analýza

✗ Časová složitost $\Theta(n^2)$

✓ Vhodné pro data malého rozsahu (desítky prvků)

- lepší nežli BubbleSort

Srovnání s SelectionSort

- SelectionSort musí vždy projít zbývající prvky pro nalezení maxima
- InsertionSort může stačit jen jediné porovnání
- ✓ výhodné pro částečně setříděné vstupy
- ✓ v průměrném případě provede cca polovinu porovnání nežli SelectionSort

Haldové třídění HeapSort

Datová struktura binární halda (binary heap)

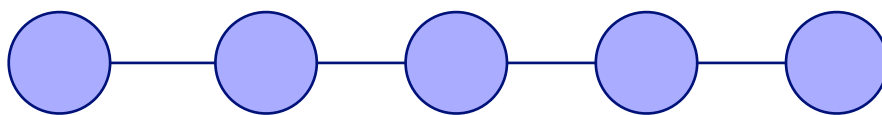
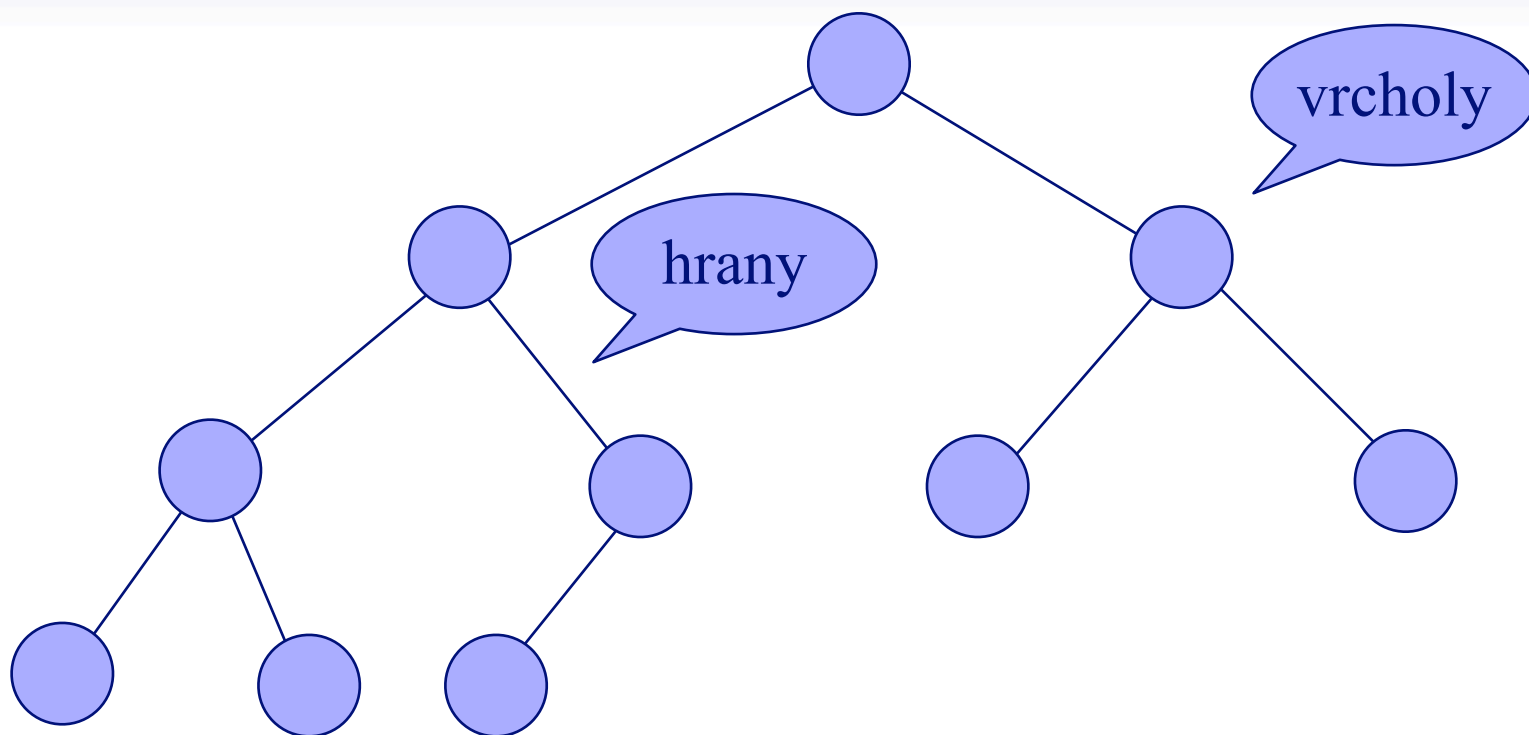
Operace

- Přidej - vložení nového prvku
- OdeberMin - odebrání minimálního prvku
- lze provést v čase $O(\log n)$
- n = počet prvků uložených v haldě

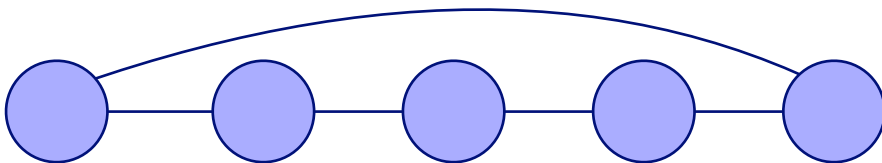
HeapSort

- z n zadaných prvků postav haldu : čas $O(n \log n)$
- n -krát odeber minimum : čas $O(n \log n)$
- třídění v čase $O(n \log n)$

Graf

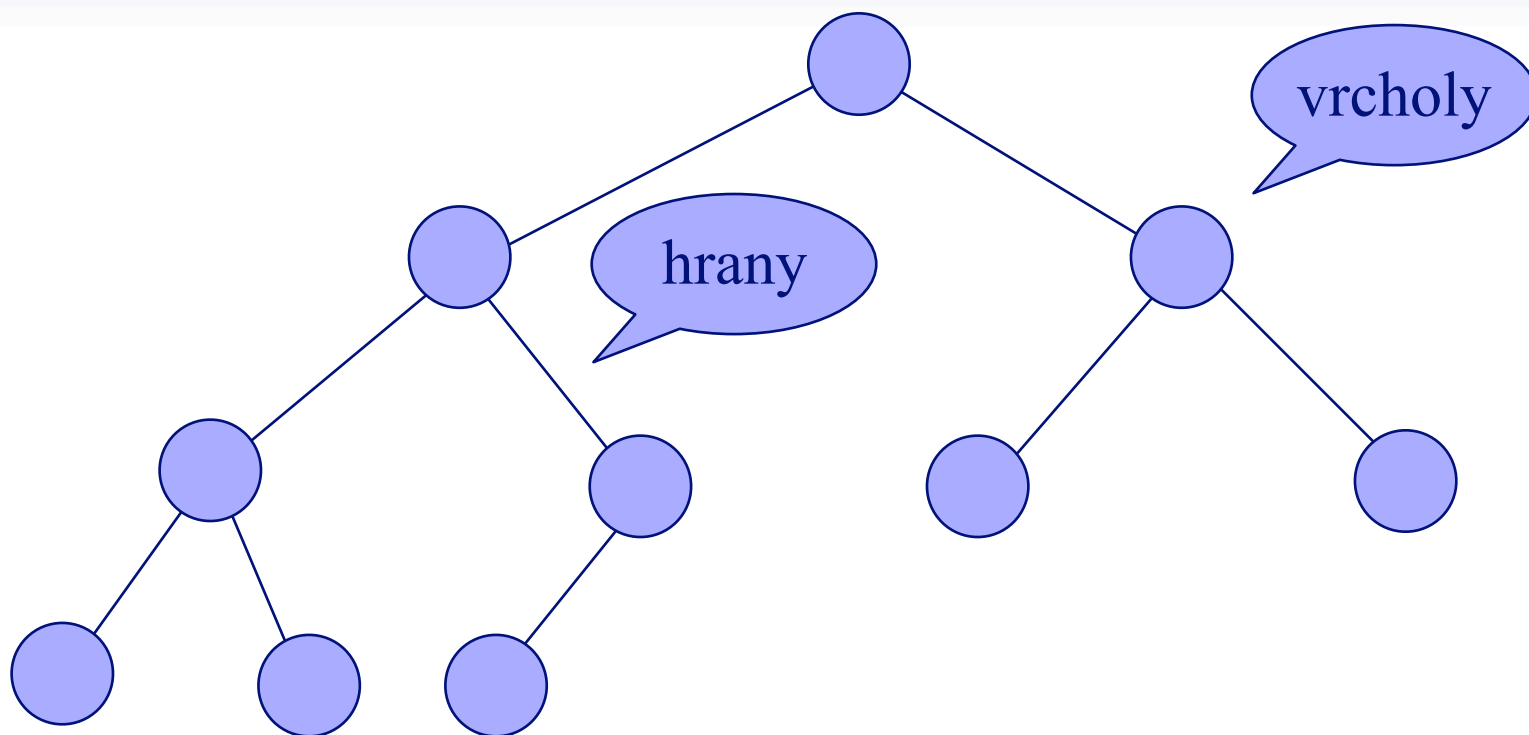


cesta (délky 4)



cyklus (délky 5)

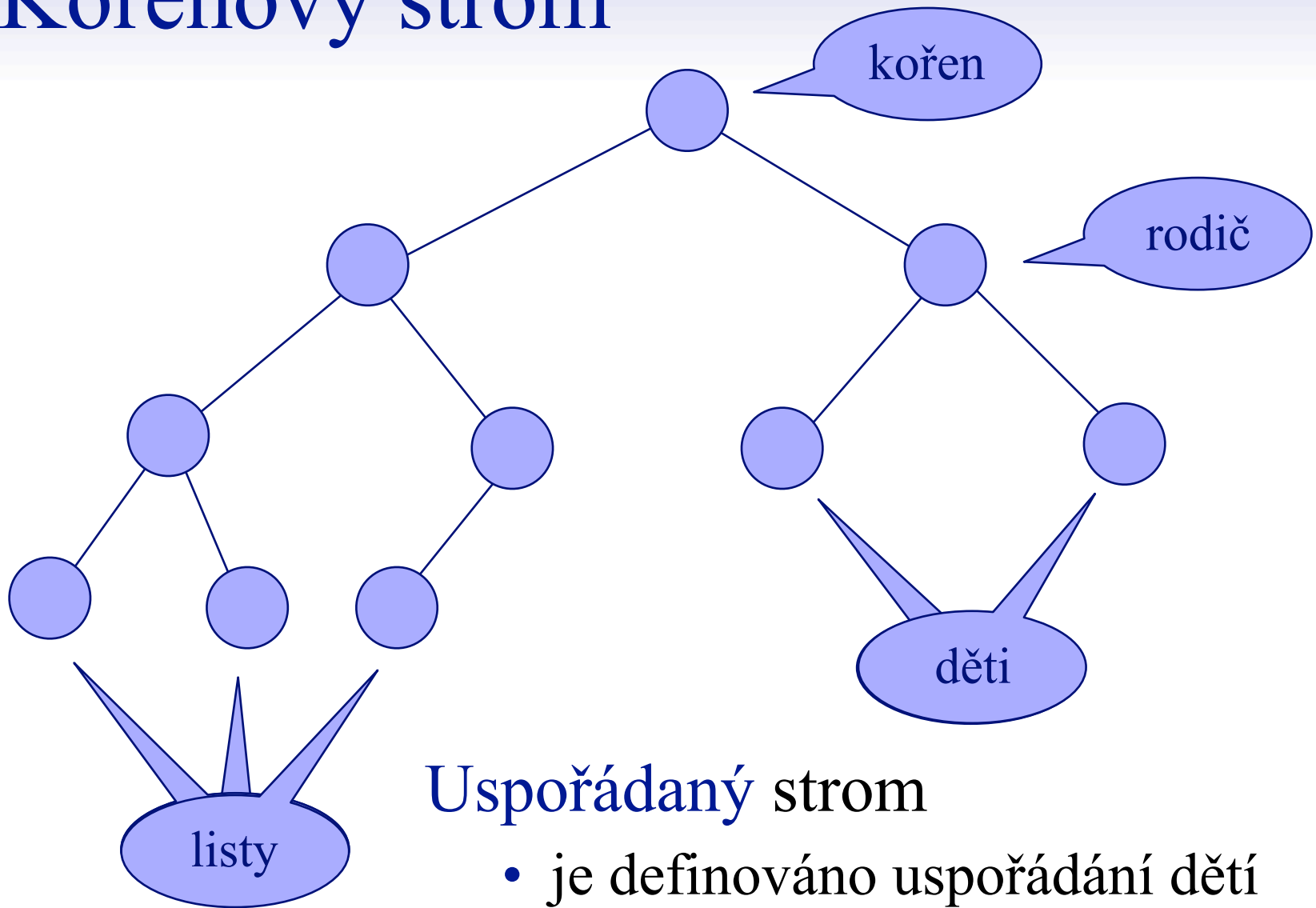
Graf



Graf je

- **souvislý** - mezi každou dvojicí vrcholů existuje cesta
- **strom** - souvislý a acyklický

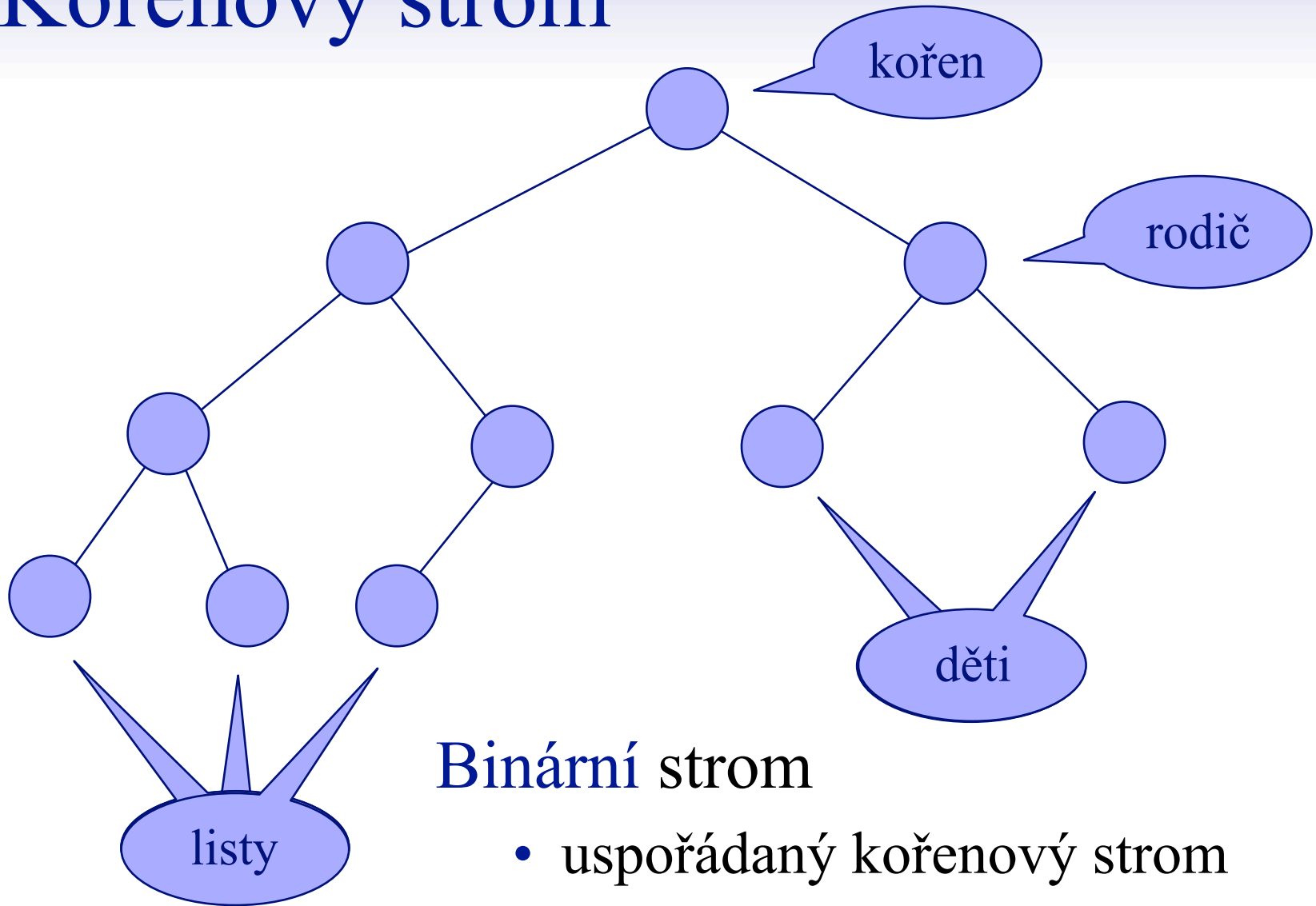
Kořenový strom



Uspořádaný strom

- je definováno uspořádání dětí
- pro každého rodiče

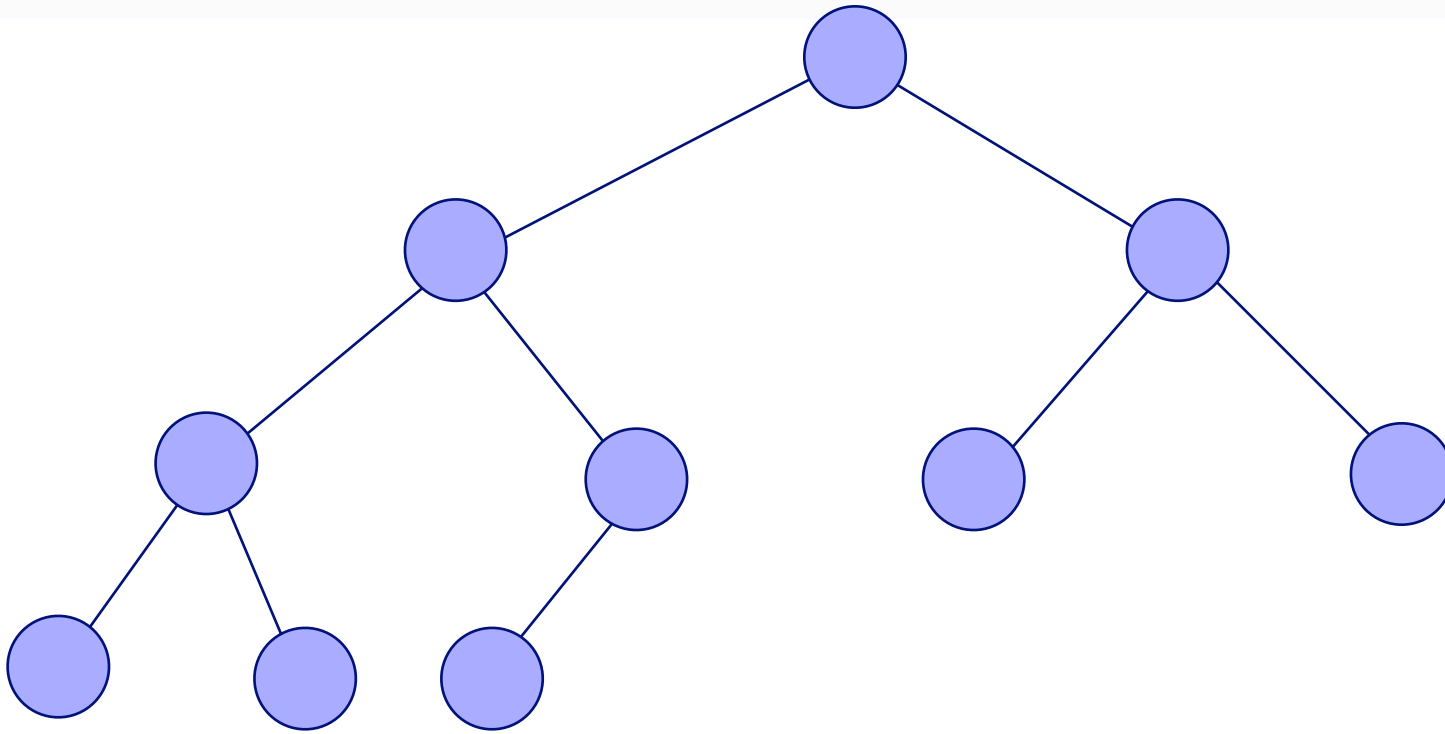
Kořenový strom



Binární strom

- uspořádaný kořenový strom
- každý rodič má nejvýše dvě děti

Kořenový strom



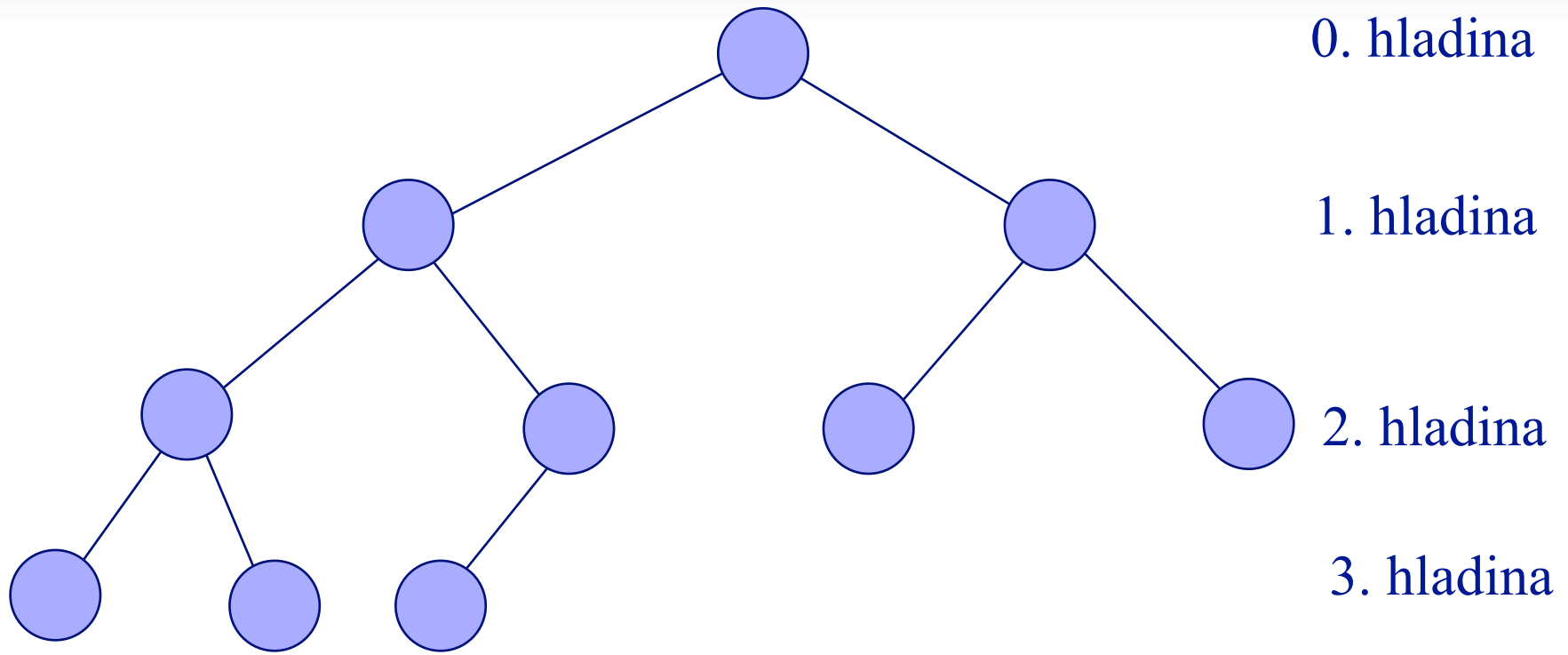
Vzdálenost vrcholů u a v

- délka nejkratší cesty mezi u a v

Výška stromu

- délka nejdelší cesty z kořene do listu

Kořenový strom

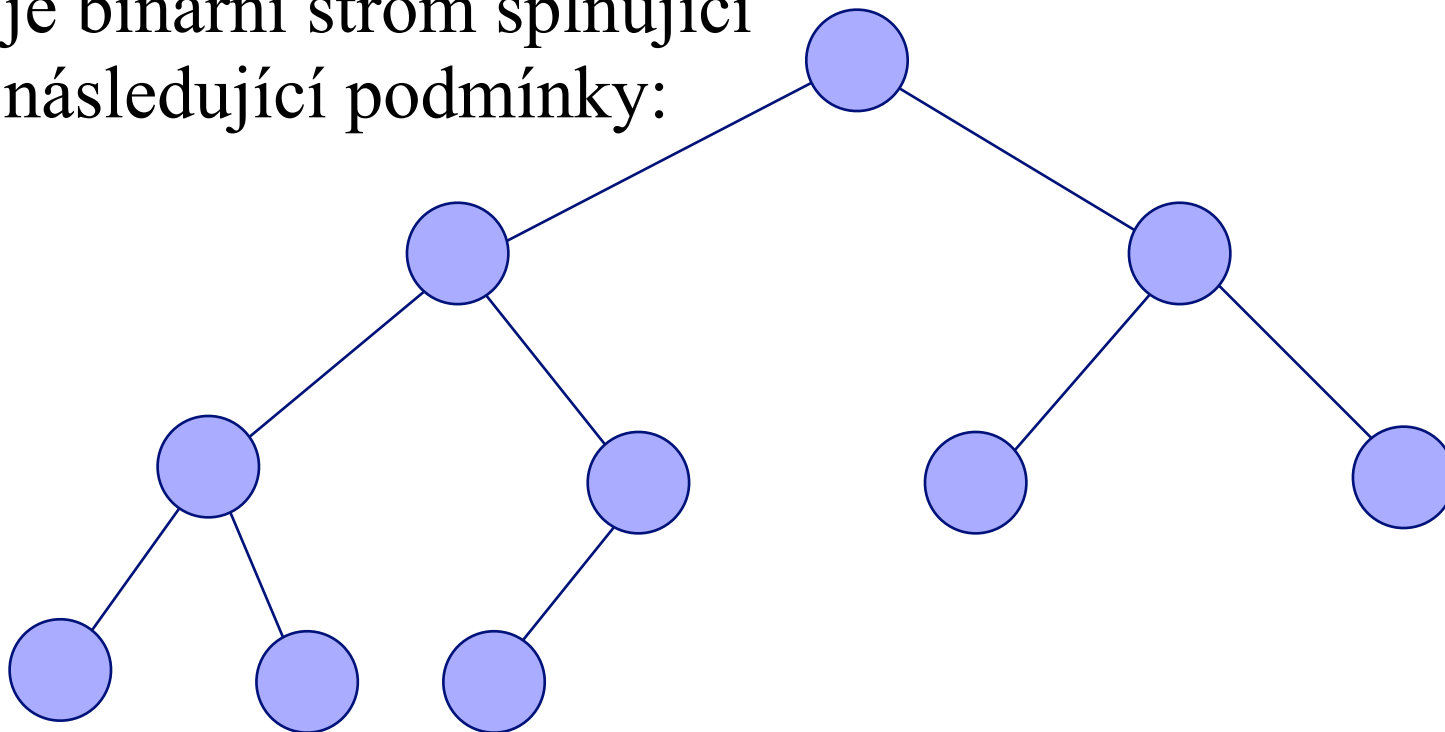


i -tá hladina

- je tvořena vrcholy ve vzdálenosti i od kořene

Binární halda

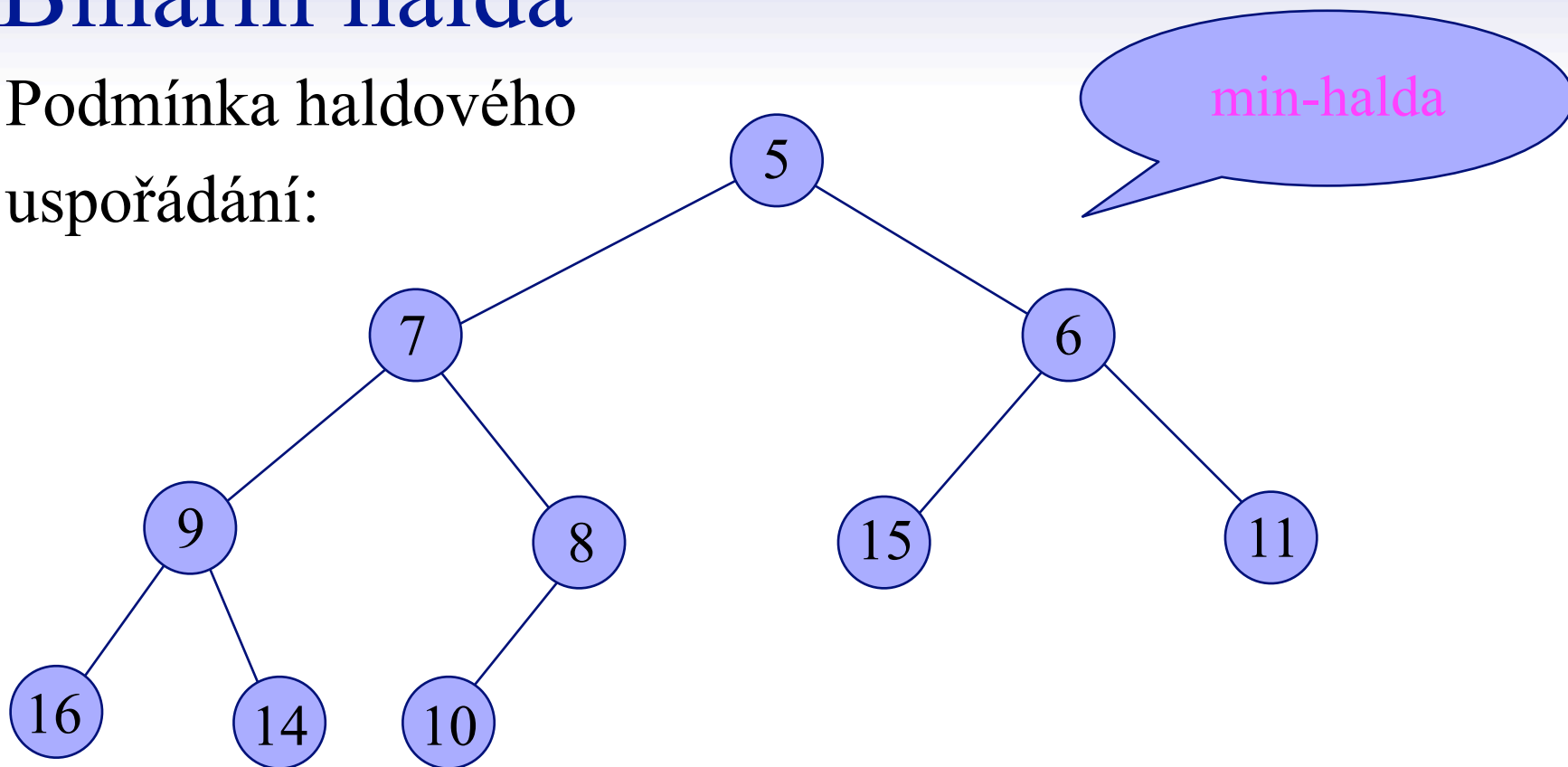
je binární strom splňující následující podmínky:



- v každé hladině od první do předposlední je max # vrcholů
- poslední hladina se zaplňuje zleva
- hodnoty uložené ve vrcholech splňují **podmínku haldového uspořádání**

Binární halda

Podmínka haldového
uspořádání:

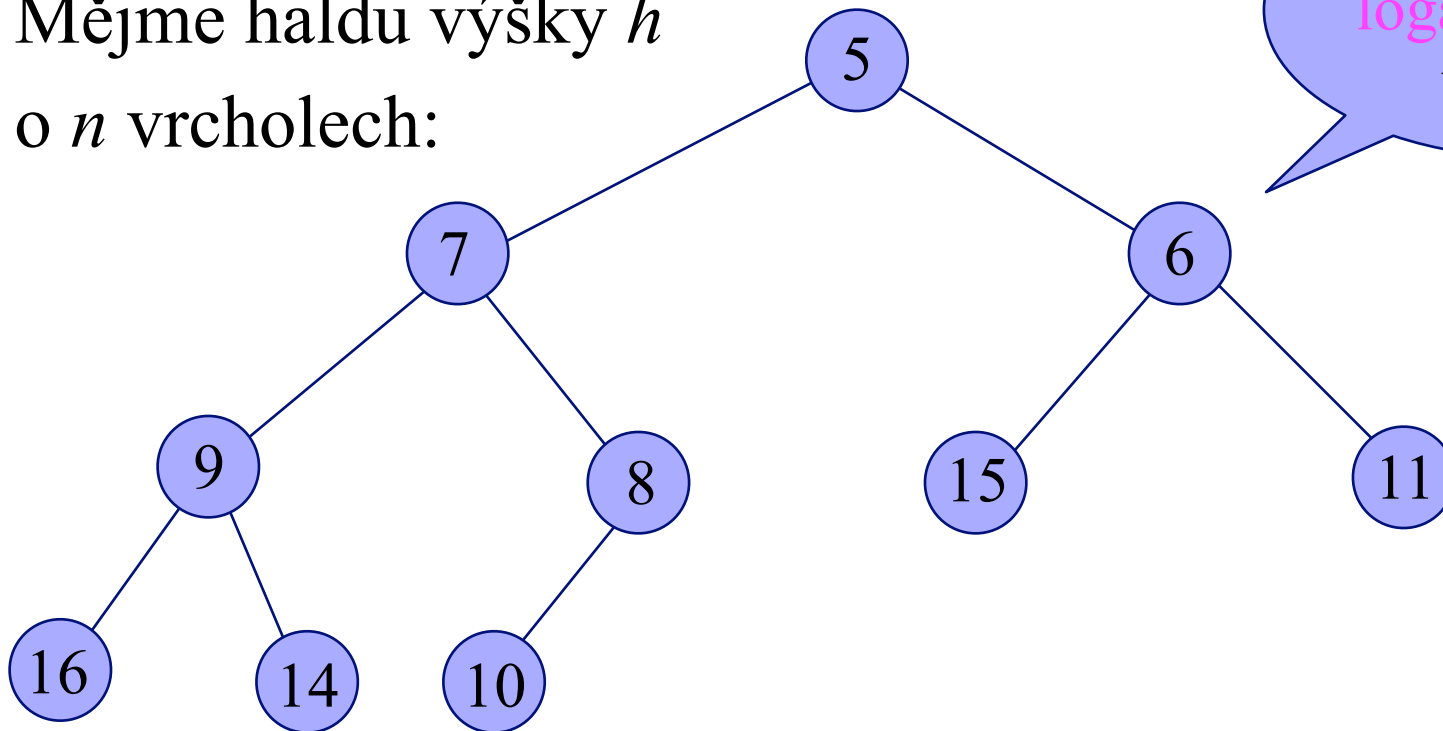


Pro každý vrchol platí, že hodnota v něm uložená je

- menší nebo rovna než hodnota v libovolném z jeho dětí (**min-halda**)
- Větší nebo rovna než hodnota v libovolném z jeho dětí (**max-halda**)

Binární halda – vlastnosti

Mějme haldu výšky h
o n vrcholech:



halda má
logaritmickou
výšku !

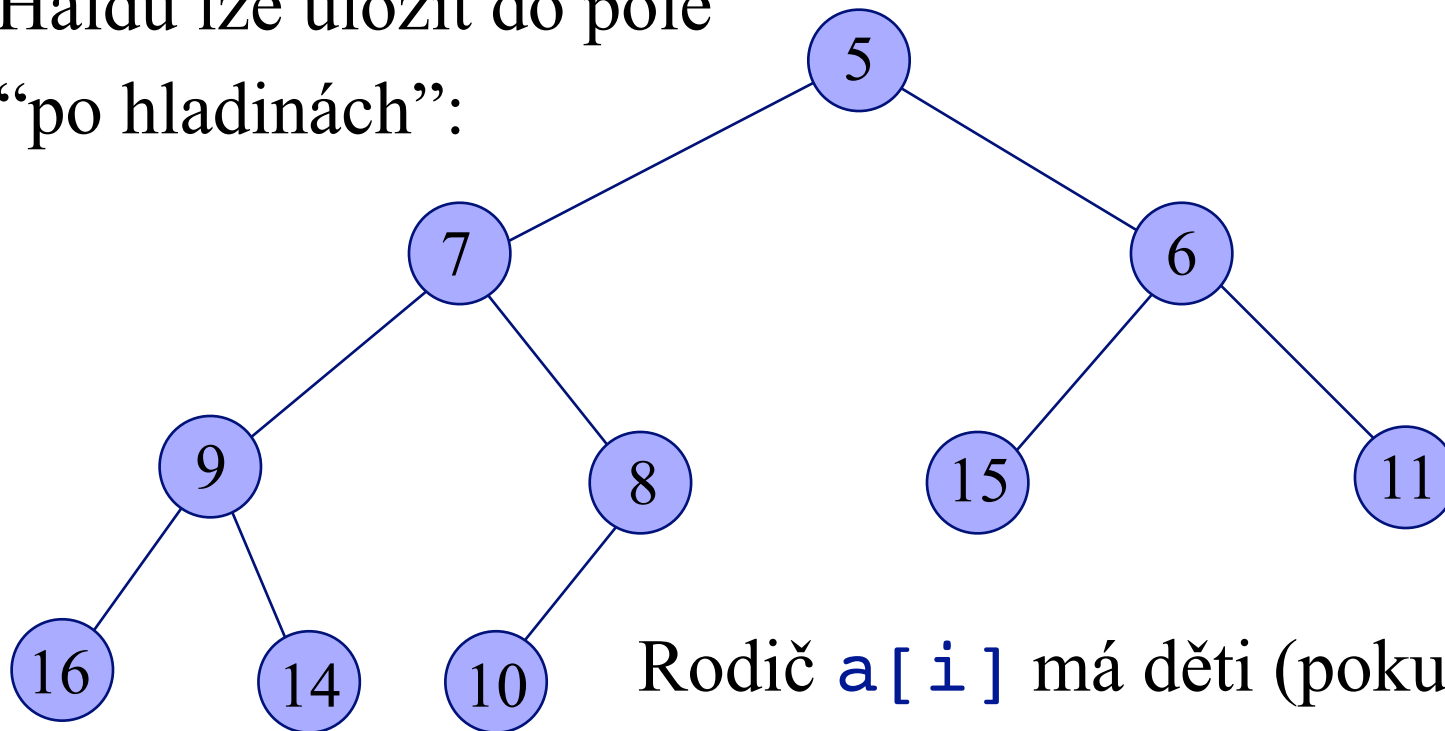
Pak platí

- na i -té hladině je 2^i vrcholů ($0 \leq i \leq h-1$)
- na poslední hladině je alespoň 1 vrchol

$$\text{Tedy } n \geq \sum_{i=0}^{h-1} 2^i + 1 = 2^h \implies h \leq \log_2 n$$

Binární halda – vlastnosti

Haldu lze uložit do pole
“po hladinách”:

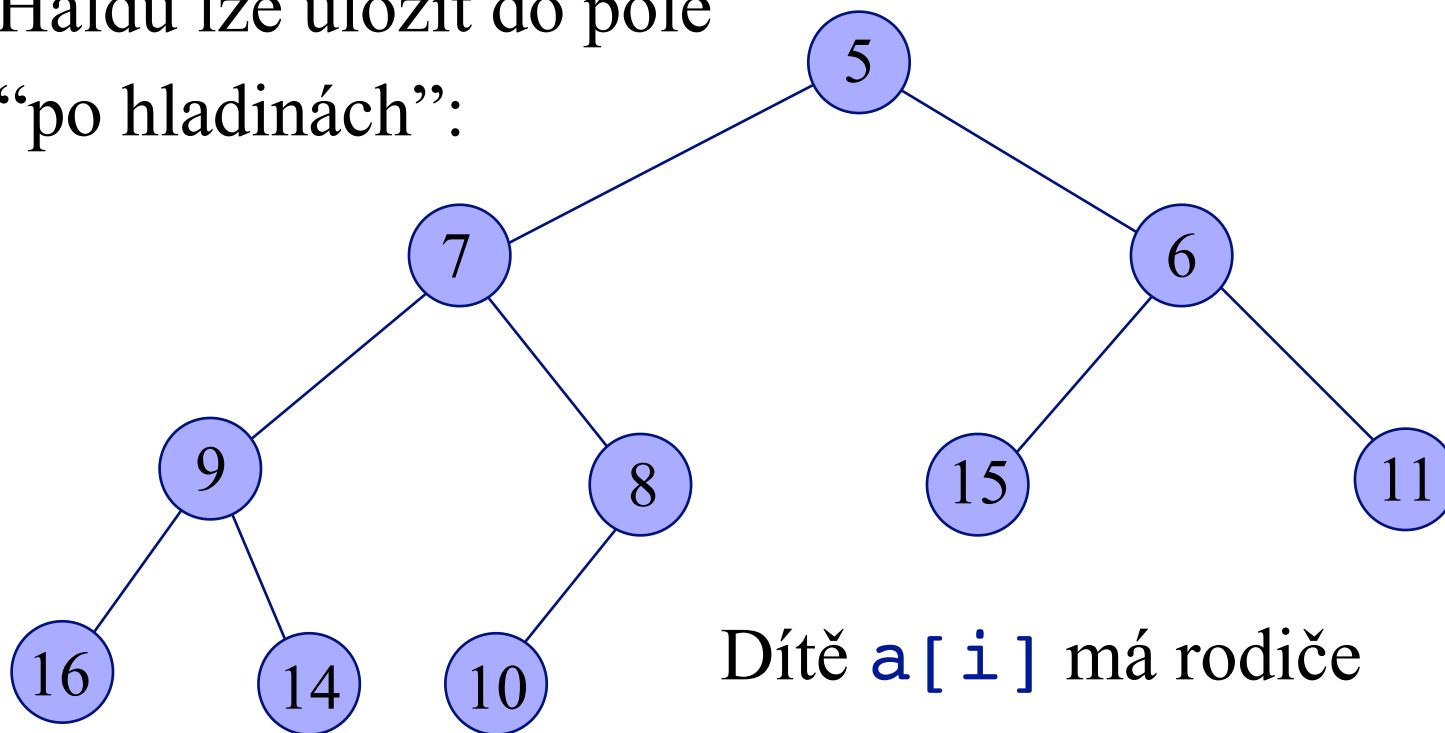


Rodič $a[i]$ má děti (pokud existují)
 $a[2*i+1]$ a $a[2*i+2]$

0	1	2	3	4	5	6	7	8	9
5	7	6	9	8	15	11	16	14	10

Binární halda – vlastnosti

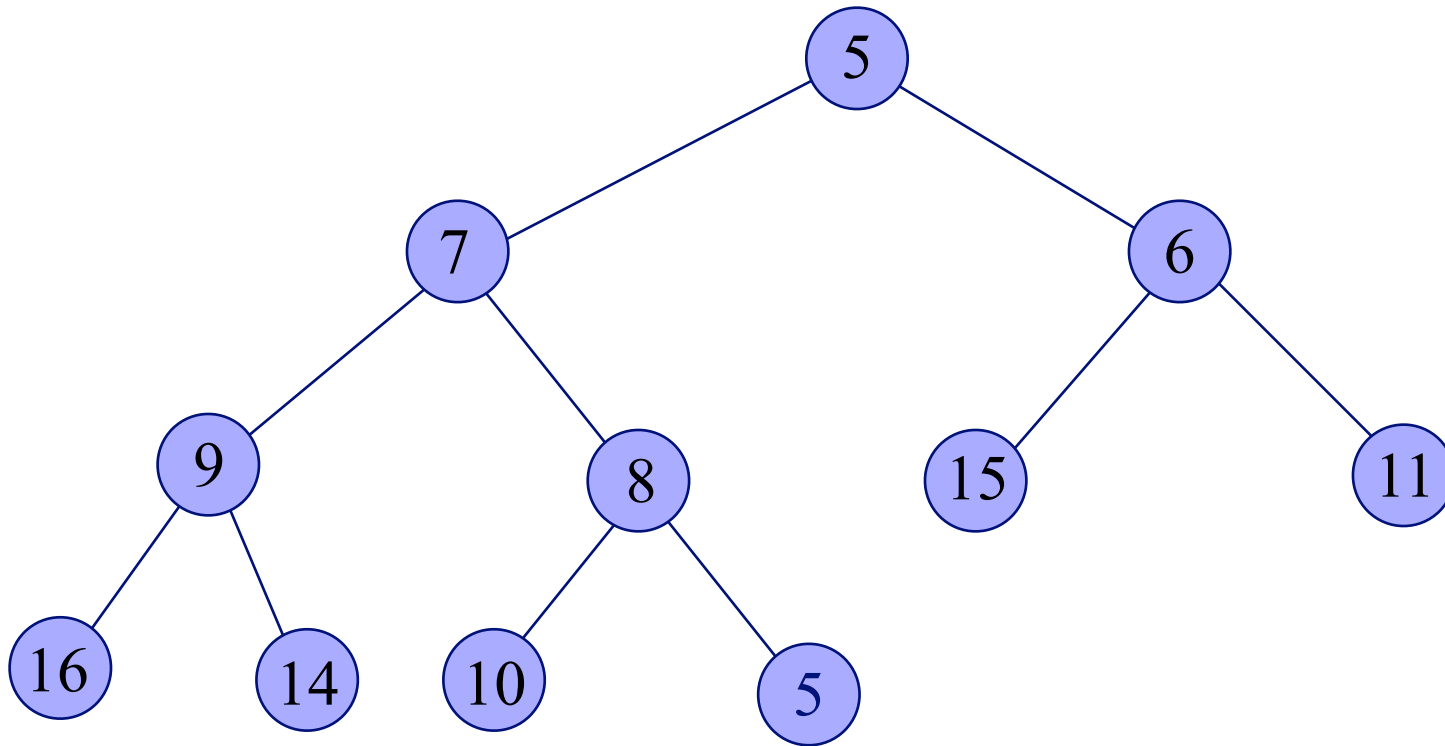
Haldu lze uložit do pole
“po hladinách”:



Dítě $a[i]$ má rodiče
 $a[(i-1)//2]$

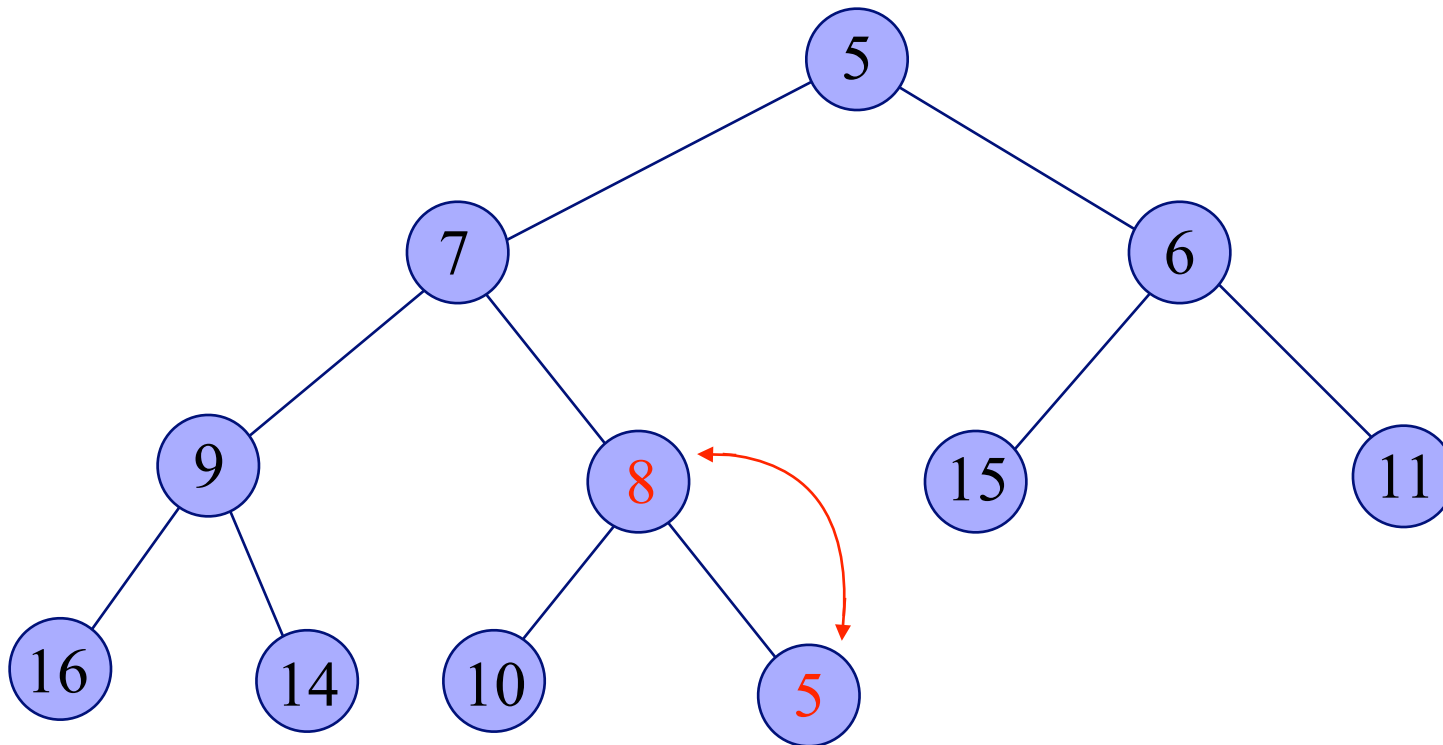
0	1	2	3	4	5	6	7	8	9
5	7	6	9	8	15	11	16	14	10

Binární halda – Přidej



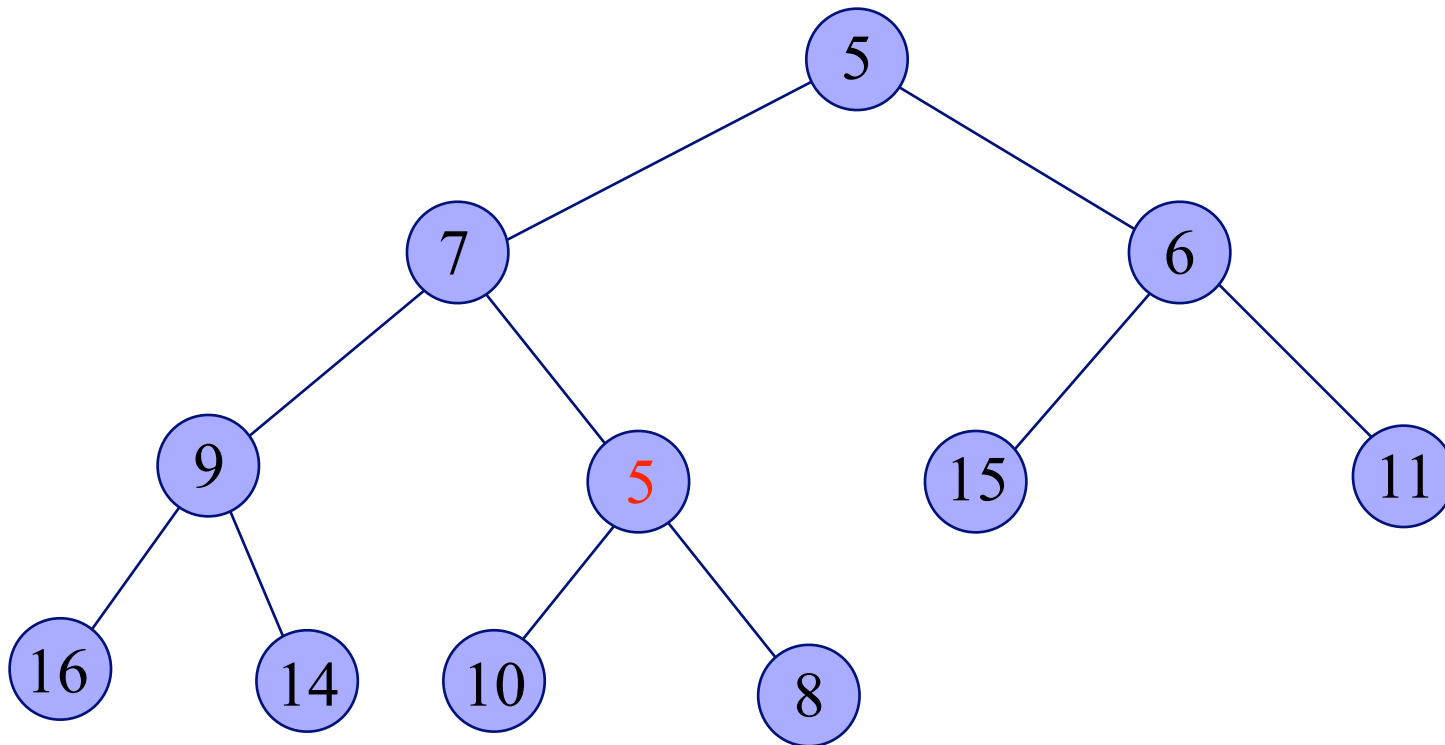
0	1	2	3	4	5	6	7	8	9	10
5	7	6	9	8	15	11	16	14	10	5

Binární halda – Přidej



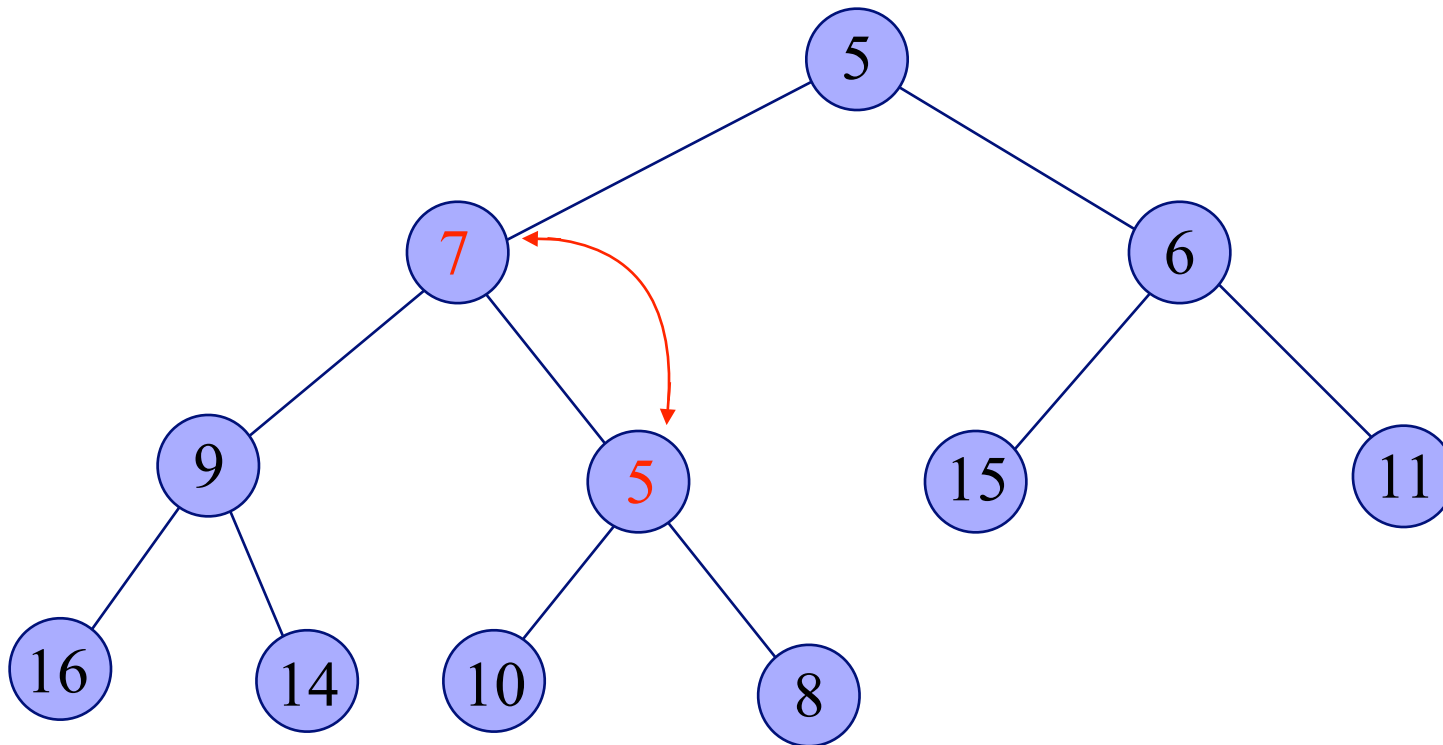
0	1	2	3	4	5	6	7	8	9	10
5	7	6	9	8	15	11	16	14	10	5

Binární halda – Přidej



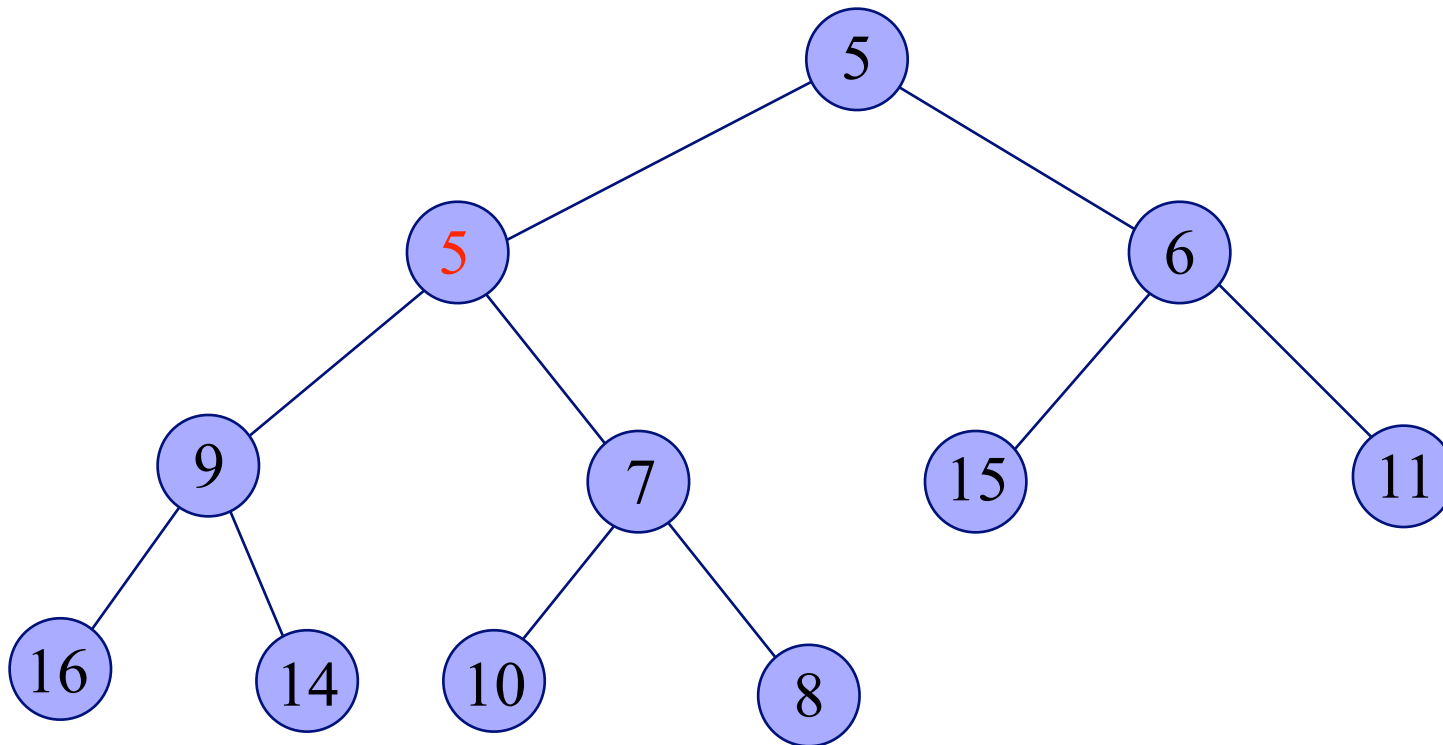
0	1	2	3	4	5	6	7	8	9	10
5	7	6	9	5	15	11	16	14	10	8

Binární halda – Přidej



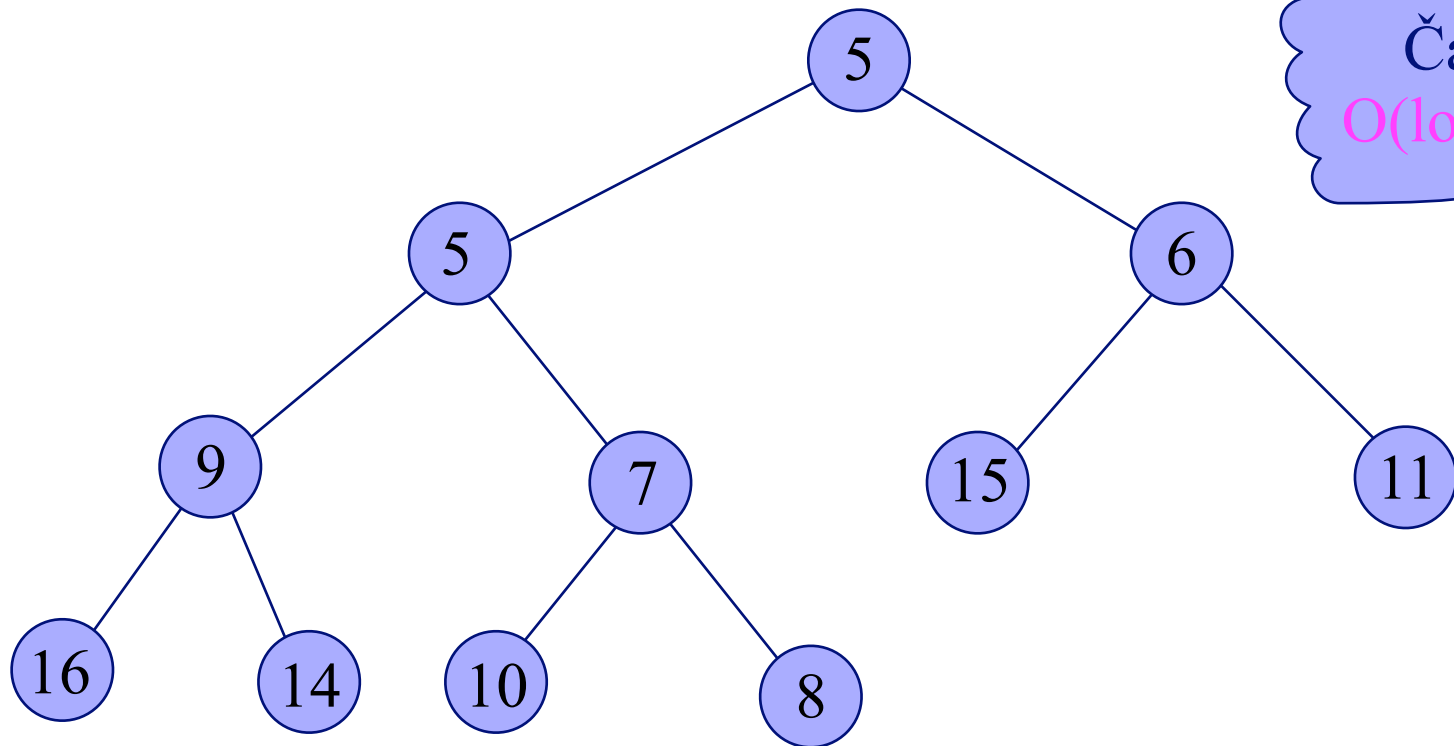
0	1	2	3	4	5	6	7	8	9	10
5	7	6	9	5	15	11	16	14	10	8

Binární halda – Přidej



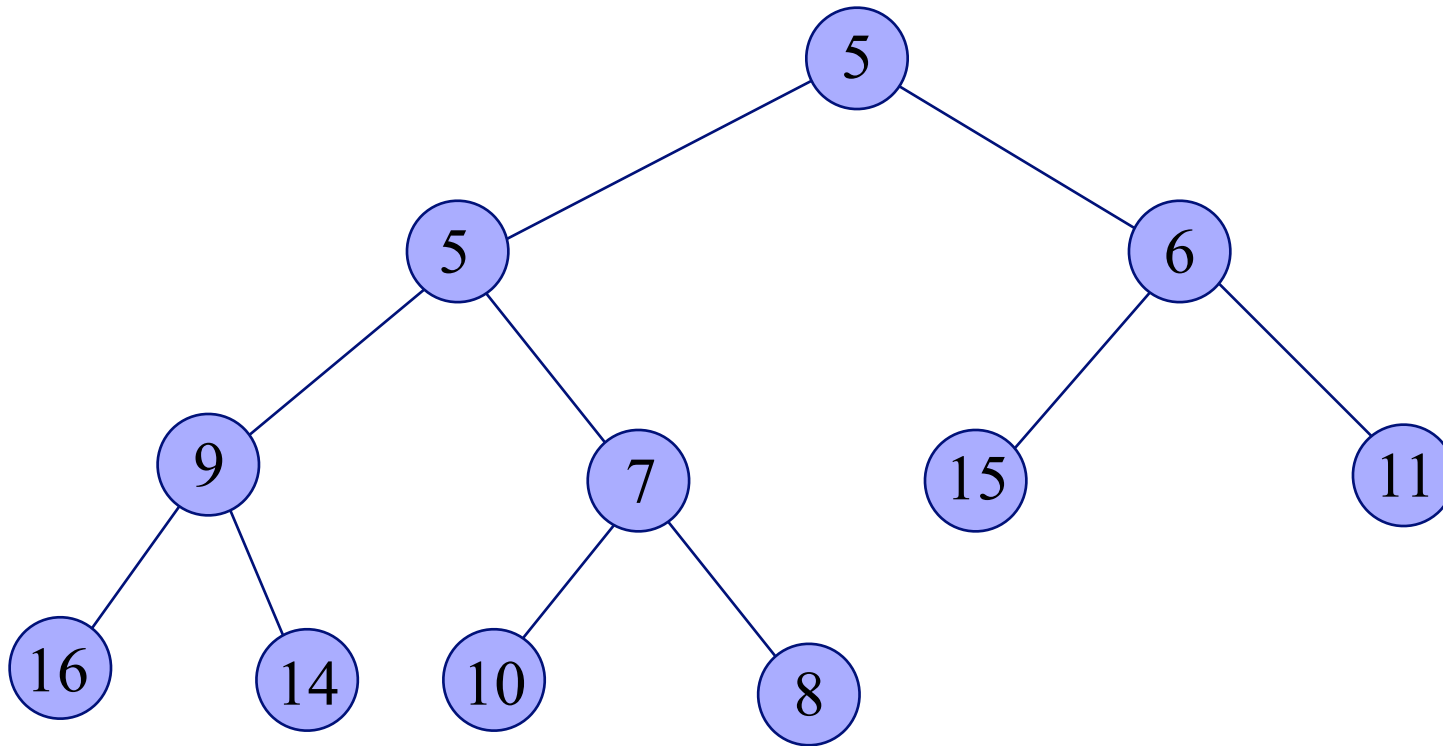
0	1	2	3	4	5	6	7	8	9	10
5	5	6	9	7	15	11	16	14	10	8

Binární halda – Přidej



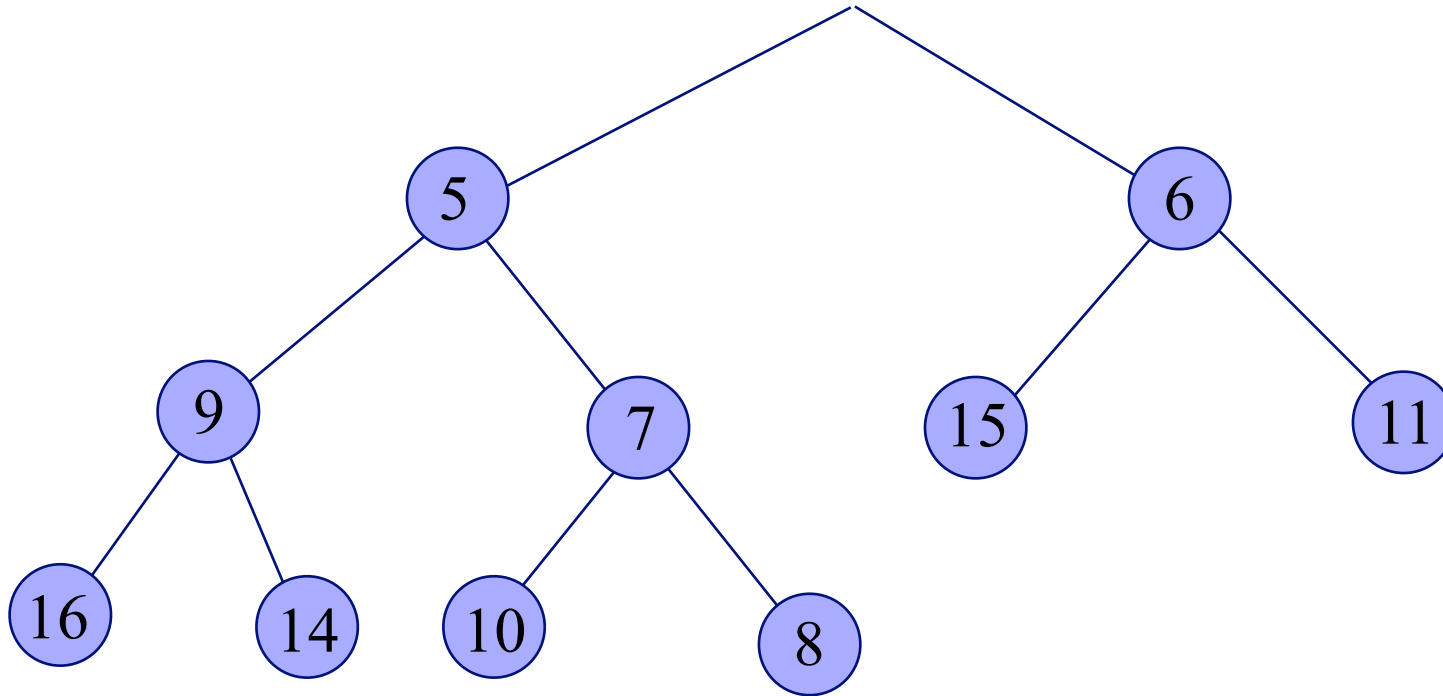
0	1	2	3	4	5	6	7	8	9	10
5	5	6	9	7	15	11	16	14	10	8

Binární halda – OdeberMin



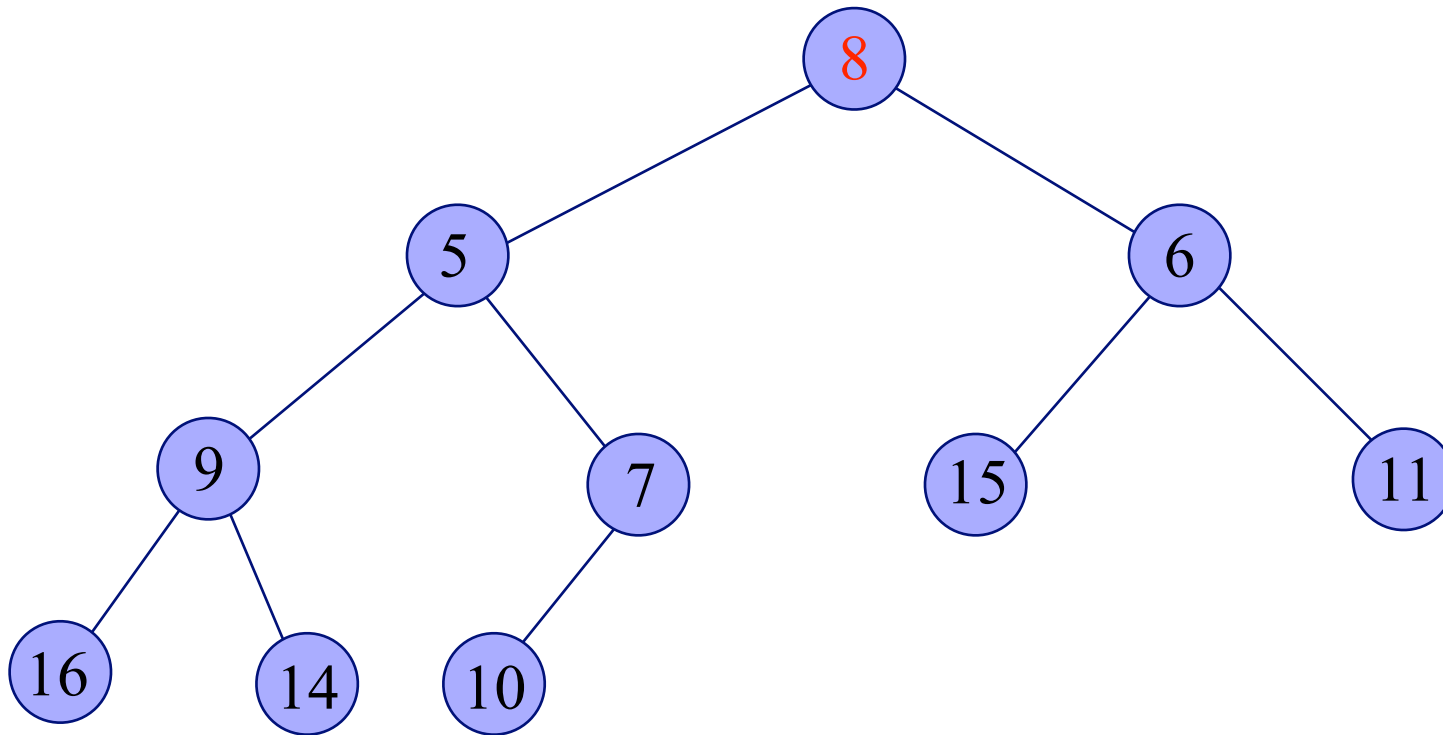
0	1	2	3	4	5	6	7	8	9	10
5	5	6	9	7	15	11	16	14	10	8

Binární halda – OdeberMin



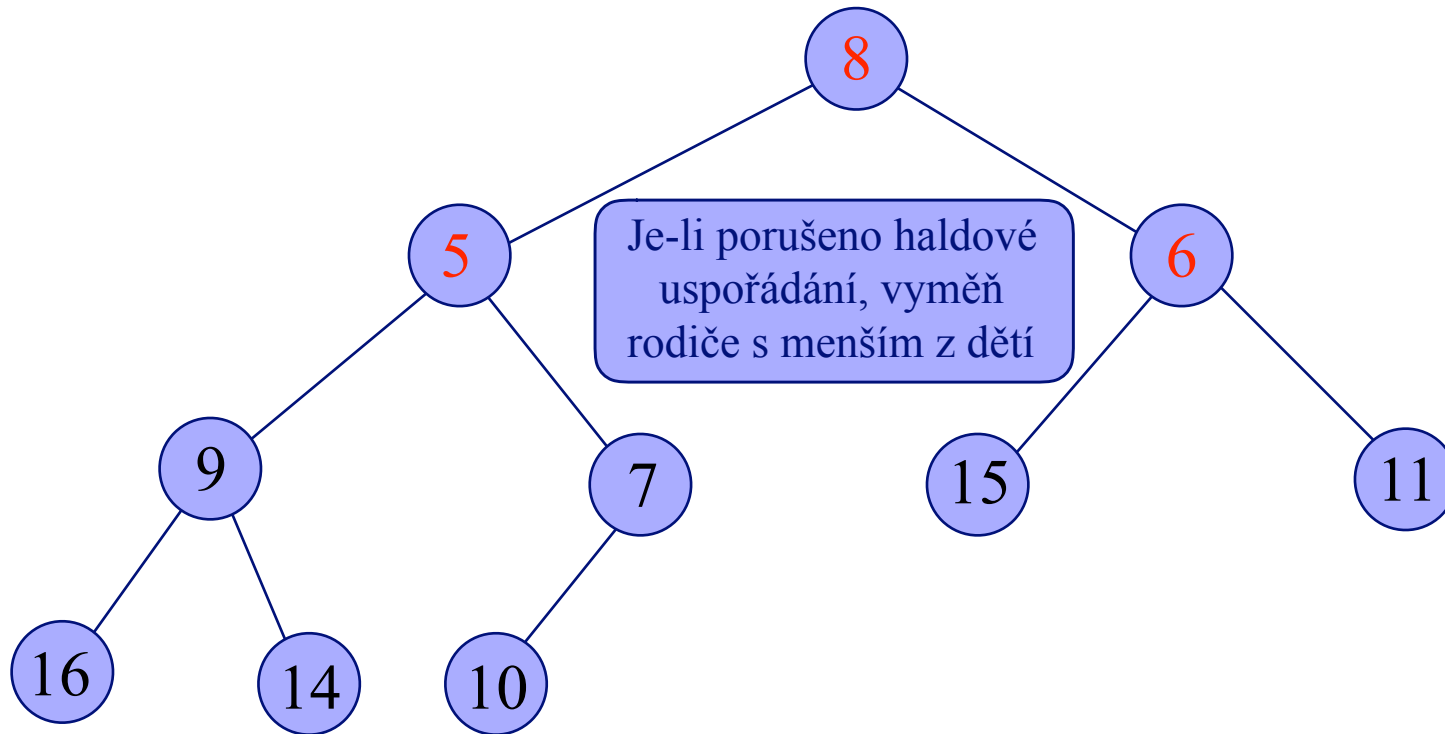
0	1	2	3	4	5	6	7	8	9	10
	5	6	9	7	15	11	16	14	10	8

Binární halda – OdeberMin



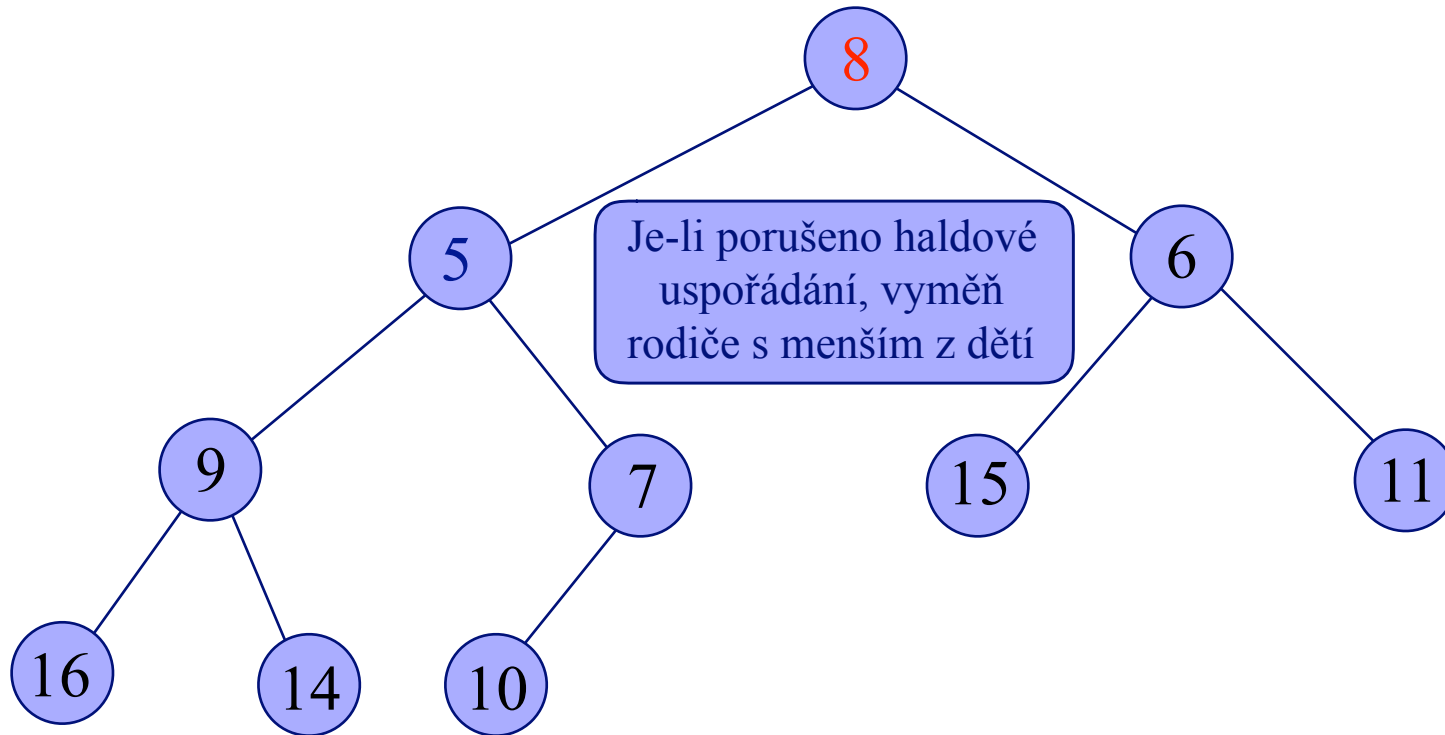
0	1	2	3	4	5	6	7	8	9	10
8	5	6	9	7	15	11	16	14	10	

Binární halda – OdeberMin



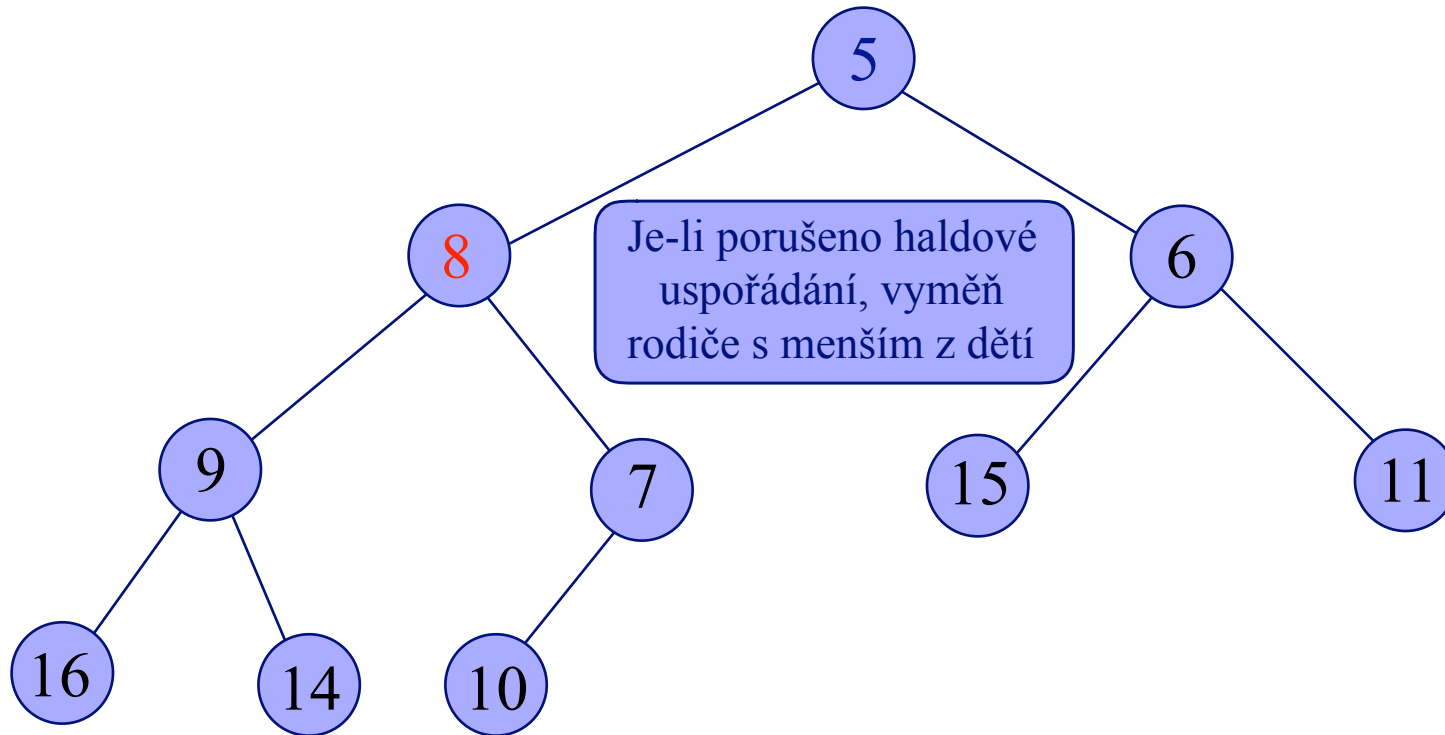
0	1	2	3	4	5	6	7	8	9	10
8	5	6	9	7	15	11	16	14	10	

Binární halda – OdeberMin



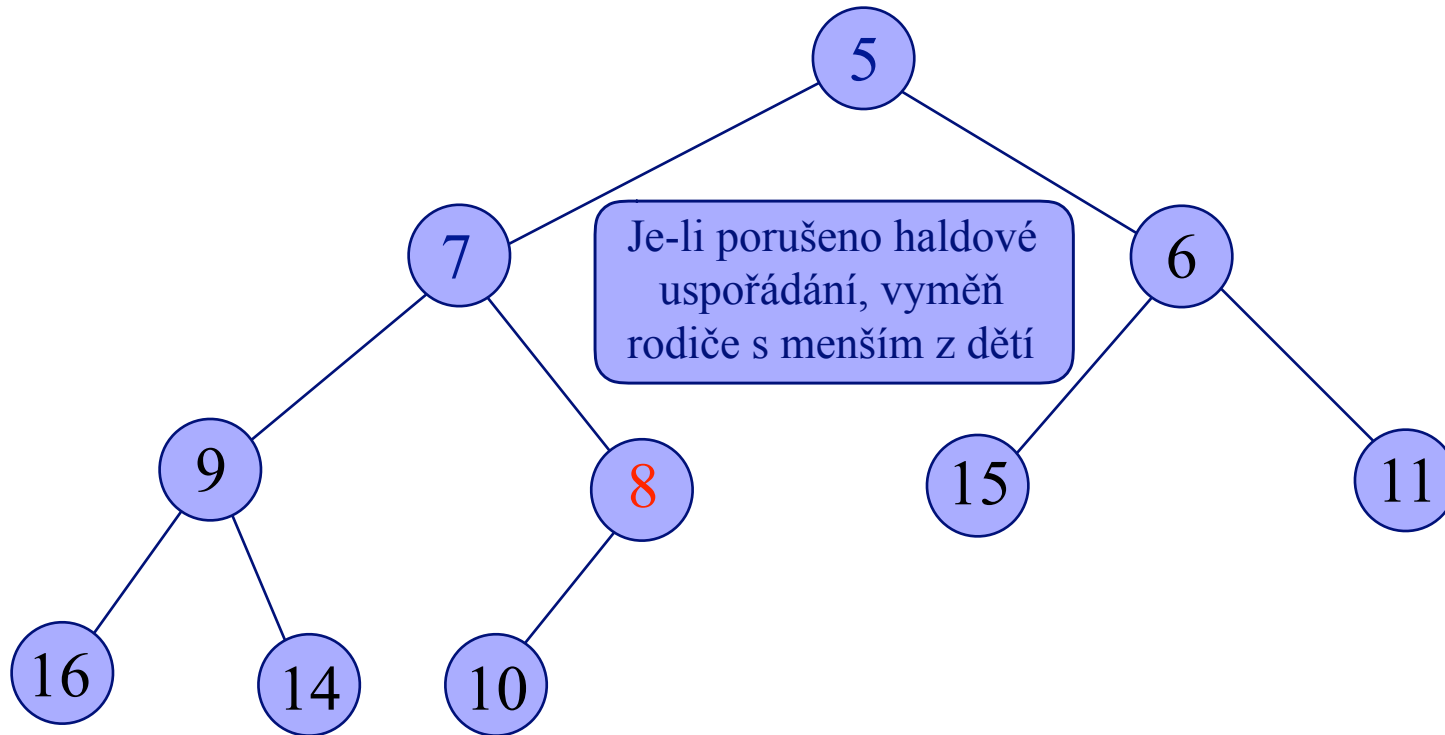
0	1	2	3	4	5	6	7	8	9	10
8	5	6	9	7	15	11	16	14	10	

Binární halda – OdeberMin



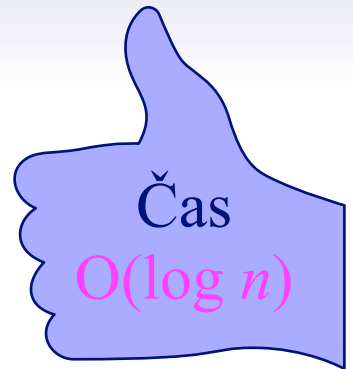
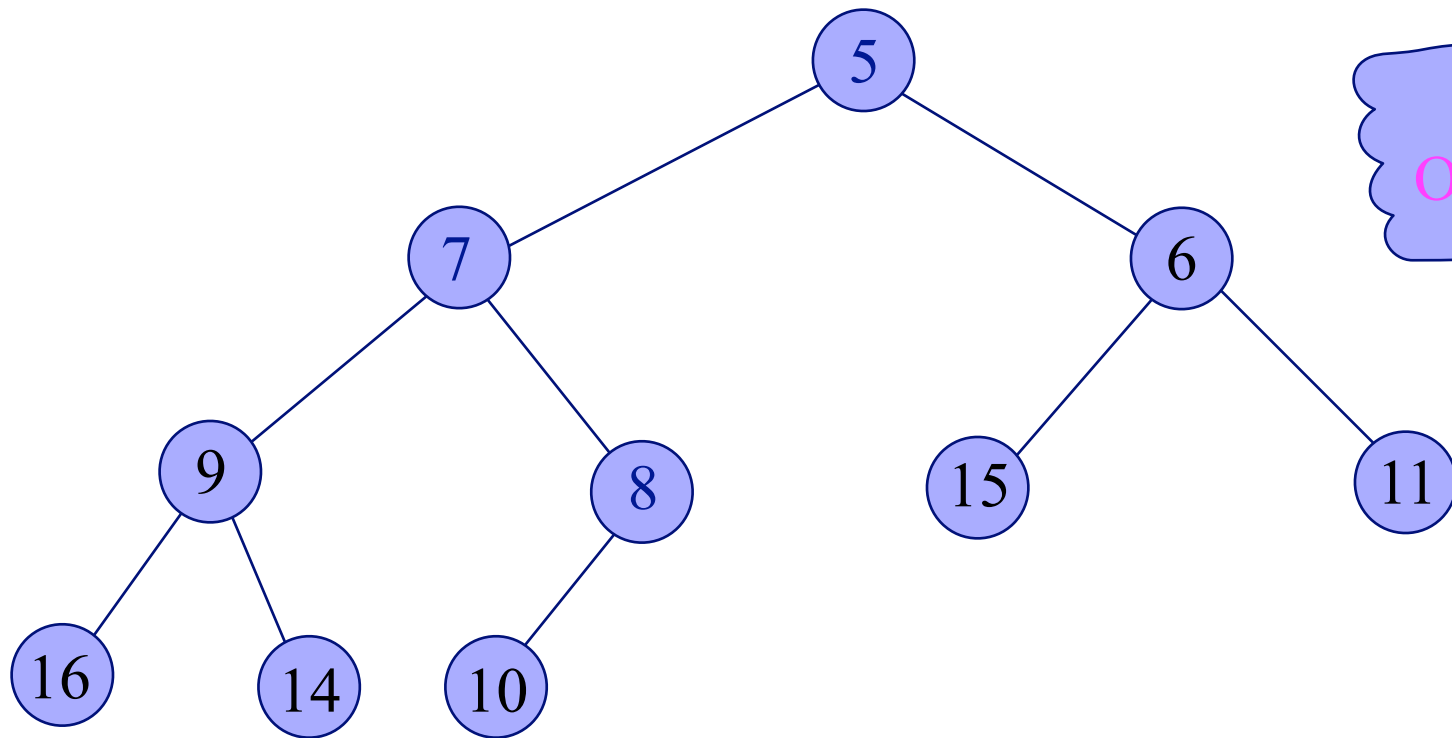
0	1	2	3	4	5	6	7	8	9	10
5	8	6	9	7	15	11	16	14	10	

Binární halda – OdeberMin



0	1	2	3	4	5	6	7	8	9	10
5	7	6	9	8	15	11	16	14	10	

Binární halda – OdeberMin



0	1	2	3	4	5	6	7	8	9	10
5	7	6	9	8	15	11	16	14	10	

Haldové třídění

Vstup: pole **a**



① Z prvků pole **a** vybuduj haldu

- začni s triviální haldou obsahující jen **a[0]**
- postupně vkládej **a[1]**, **a[2]**,... pomocí **Přidej**
- v poli **a** je nyní uložena halda

② Z haldy postupně odebírej minima

- pomocí **OdeberMin**

Problém

pouze konstantní
pracovní paměť

- jak zajistíme **třídění na místě (in situ)** ?

Problémy

① V jazyce Python sestavte funkci

`heapSort(a)`

která setřídí prvky zadaného pole `a` vzestupně haldovým tříděním. Váš algoritmus by měl třídit na místě, tj. může využívat jen konstantní pracovní paměť.

Dolní odhad časové složitosti třídění

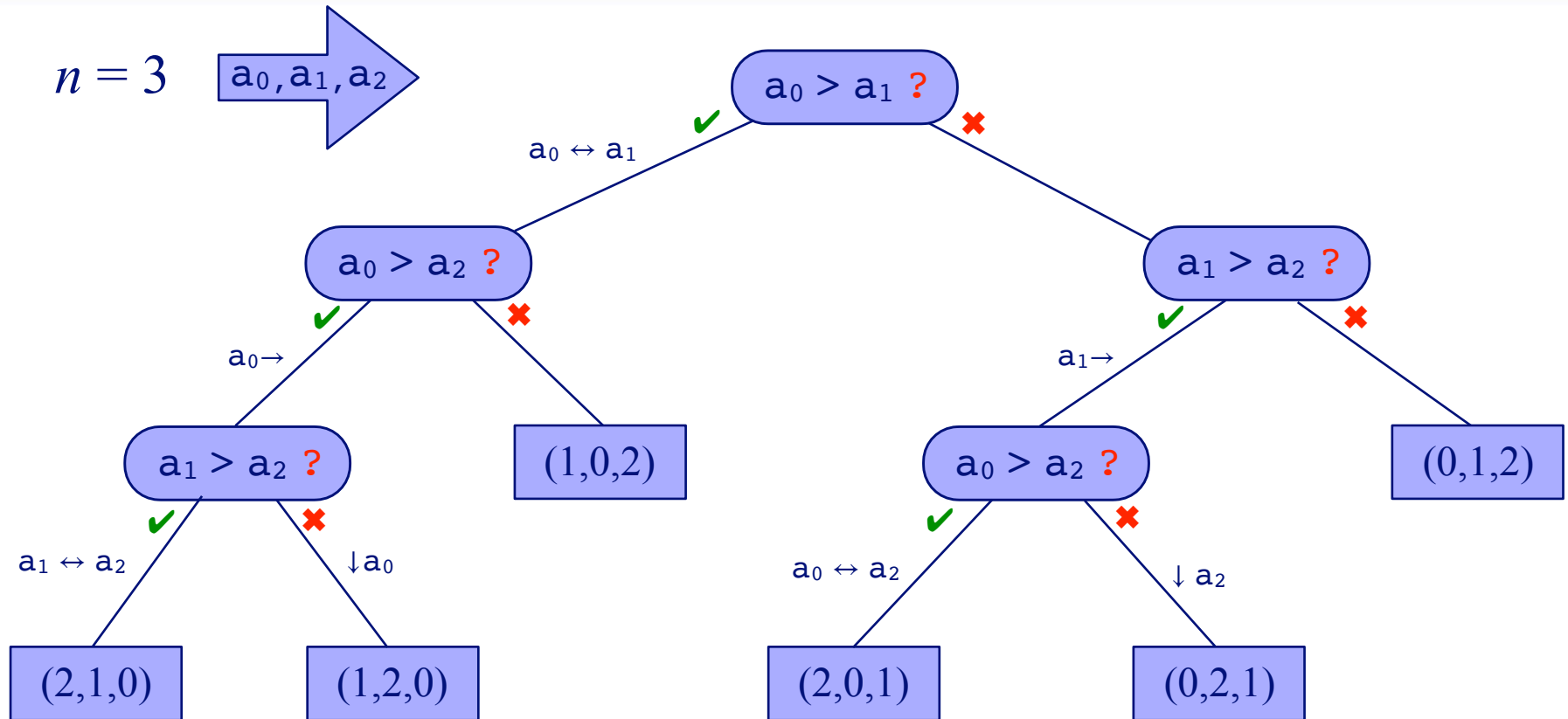
Porovnávací třídící algoritmus

- třídí na základě porovnávání dvojic prvků
 - » $a_i < a_j$, $a_i > a_j$, $a_i \leq a_j$, $a_i \geq a_j$, $a_i = a_j$
- hodnoty tříděných prvků nevyužívá


Rozhodovací strom

- reprezentuje průběh porovnání prováděných
- (konkrétním) porovnávacím algoritmem
- nad vstupem délky n
- vnitřní vrcholy – porovnání
- listy – koncové stavy výpočtu

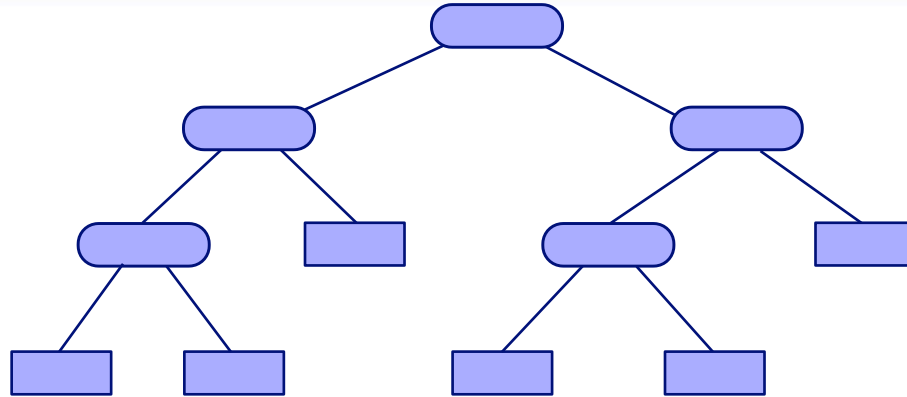
Rozhodovací strom pro InsertionSort



listů = # permutací množiny $\{0,1,2\} = 3!$

 **Zobecnění:** Každý rozhodovací strom porovnávacího algoritmu nad vstupem délky n má alespoň $n!$ listů.

Rozhodovací stromy – vlastnosti



Cesta z kořene do listu = výpočet

- délka cesty = # porovnání

Maximální # porovnání nad vstupem délky n

- = výška rozhodovacího stromu

listů binárního stromu výšky h je $\leq 2^h$

- # listů rozhodovacího stromu $\geq n!$

👉 $2^h \geq n!$ 👉 $h \geq \log_2(n!)$

Hrátky s funkcí $n!$

$$\begin{aligned} n! &= \sqrt{n!} \cdot \sqrt{n!} = \sqrt{n \cdot (n-1) \cdot \dots \cdot 1} \cdot \sqrt{1 \cdot 2 \cdot \dots \cdot n} \\ &= \sqrt{n \cdot 1} \cdot \sqrt{(n-1) \cdot 2 \cdot \dots} \cdot \sqrt{1 \cdot n} \end{aligned}$$

Pro $0 \leq k \leq n-1$ platí:

$$\begin{aligned} (n-k)(k+1) &= nk + n - k^2 - k \\ &= n + k(n-1-k) \geq n \end{aligned}$$

$$n! \geq (\sqrt{n})^n = n^{n/2}$$

$$\log_2(n!) \geq \log_2(n^{n/2}) = \frac{1}{2} \cdot n \cdot \log_2 n$$

 **Maximální počet porovnání** nad vstupem délky n je
 $\geq \text{konstanta} \cdot n \cdot \log_2 n$

Složitost problému třídění

- ☝ **Závěr:** Každý porovnávací třídící algoritmus provede v nejhorším případě $\Omega(n \log n)$ porovnání.
- ☞ **Důsledek:** Algoritmus haldového třídění má asymptoticky optimální časovou složitost.
- ☞ **Důsledek:** Složitost problému vnitřního třídění porovnávacím algoritmem je $\Theta(n \log n)$.

Problémy

② Uvažte třídící algoritmus, který pro libovolné dva prvky a_i , a_j na vstupu může na dotaz $a_i ? a_j$ obdržet jednu ze tří možných odpovědí:

- $a_i < a_j$
- $a_i > a_j$
- $a_i = a_j$

Bude náš dolní odhad platit i tomto případě?