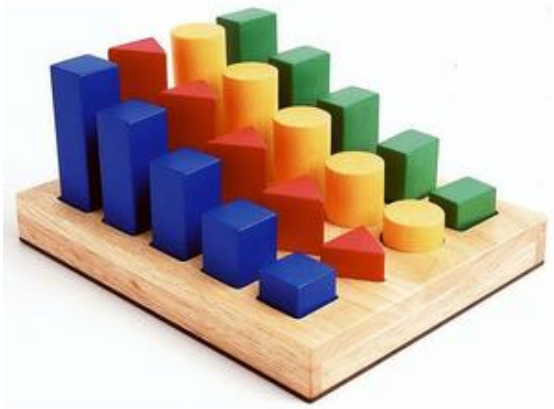
5. prednáška (15.3.2021)

Triediace algoritmy

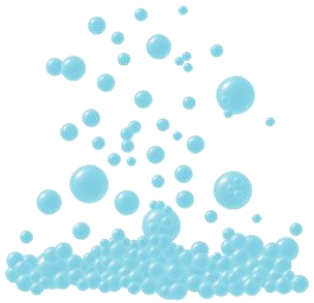
alebo

tri poučné príbehy
o troch obyčajných algoritmoch



Sumarizácia 2. prednášky

- **Binárne vyhľadávanie** v čase $O(\log n)$
  - má zmysel mať **usporiadané** prvky v postupnosti
- Ako usporiadať prvky/hodnoty postupnosti?
  - bublinkové triedenie
  - triedenie výberom
  - triedenie z 2. sady zadaní ...



- Algoritmy sa líšia prístupom k úlohe, ale všetky bežia v čase $O(n^2)$, kde n je počet prvkov, ktoré chceme utriediť (usporiadať)
  - v najhoršom prípade je každý z nich v čase $\Omega(n^2)$

Čo sa vlastne dá usporadúvať?

- **Pozorovanie** z cvičení:
  - **nezáleží** na tom, **čo usporadúvame** – stačí, ak pre ľubovoľné 2 prvky (hodnoty) vieme povedať, čo je menšie („má byť skôr v usporiadanej postupnosti“)
  - metóda `porovnaj`
    - porovná prvky na dvoch zadaných indexoch
  - metóda `vymen`
    - vzájomne vymení prvky na dvoch zadaných indexoch
- „Odborne“: vieme usporiadať prvky z **lineárne usporiadanej množiny**
    - čísla, znakové reťazce, ...

Poďme usporiadať zvieratká

Obama – the sorter

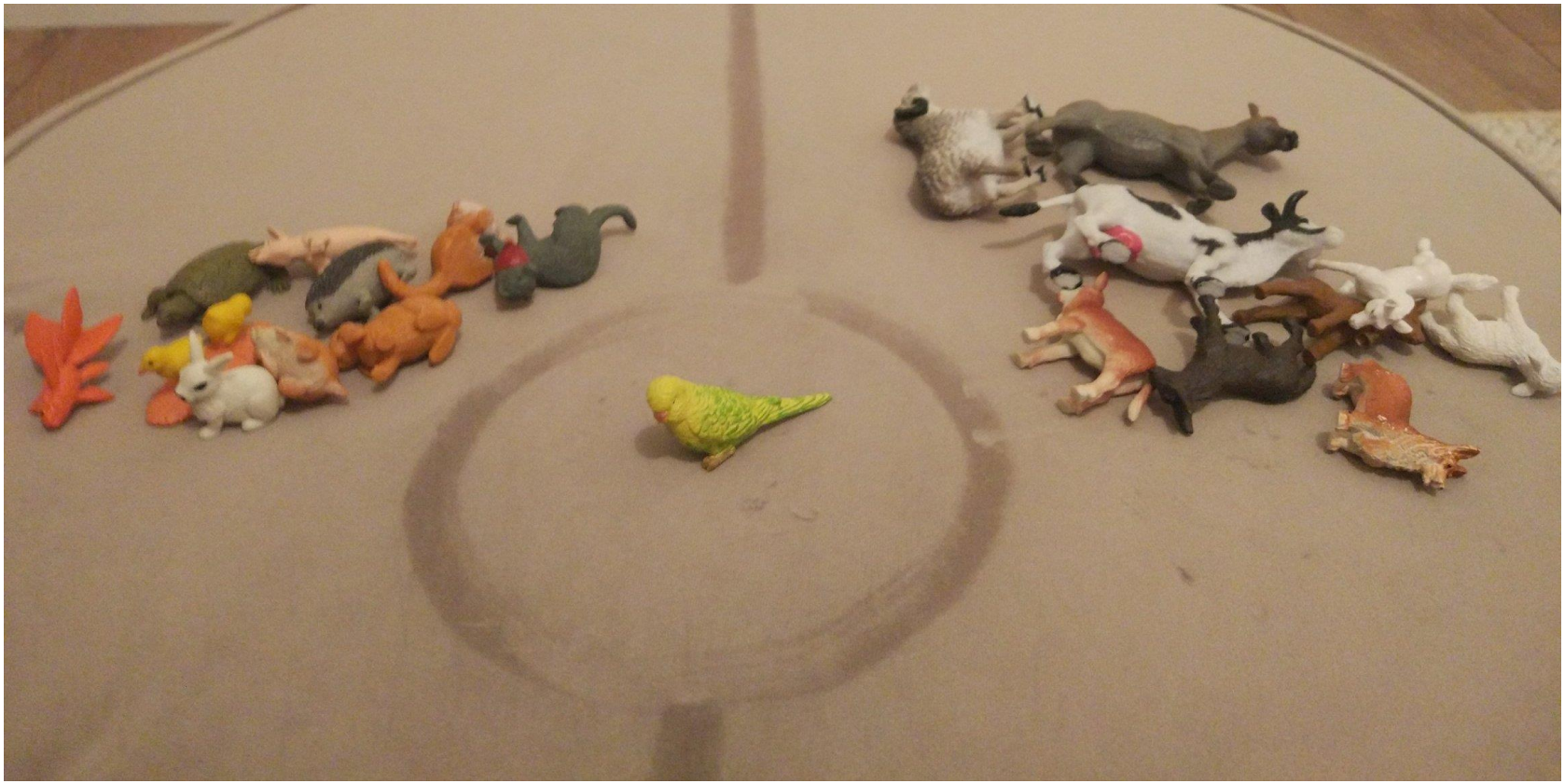
Q: What is the most efficient way to sort a million of 32-bit integers?

A: *I think the bubble sort would be the wrong way to go.*

Ako sa dá triediť lepšie než v čase $O(n^2)$?

Myšlienka č. 1

- rozdeliť si zvieratká na menšie a väčšie
- rozdeliť si problém na 2 menšie podproblémy



Myšlienka č. 1

- ... „a oddelí ľudí jedných od druhých, tak ako pastier oddeľuje ovce od capov. A postaví ovce po svojej **pravici**, ale capov po svojej **ľavici**.“ ...

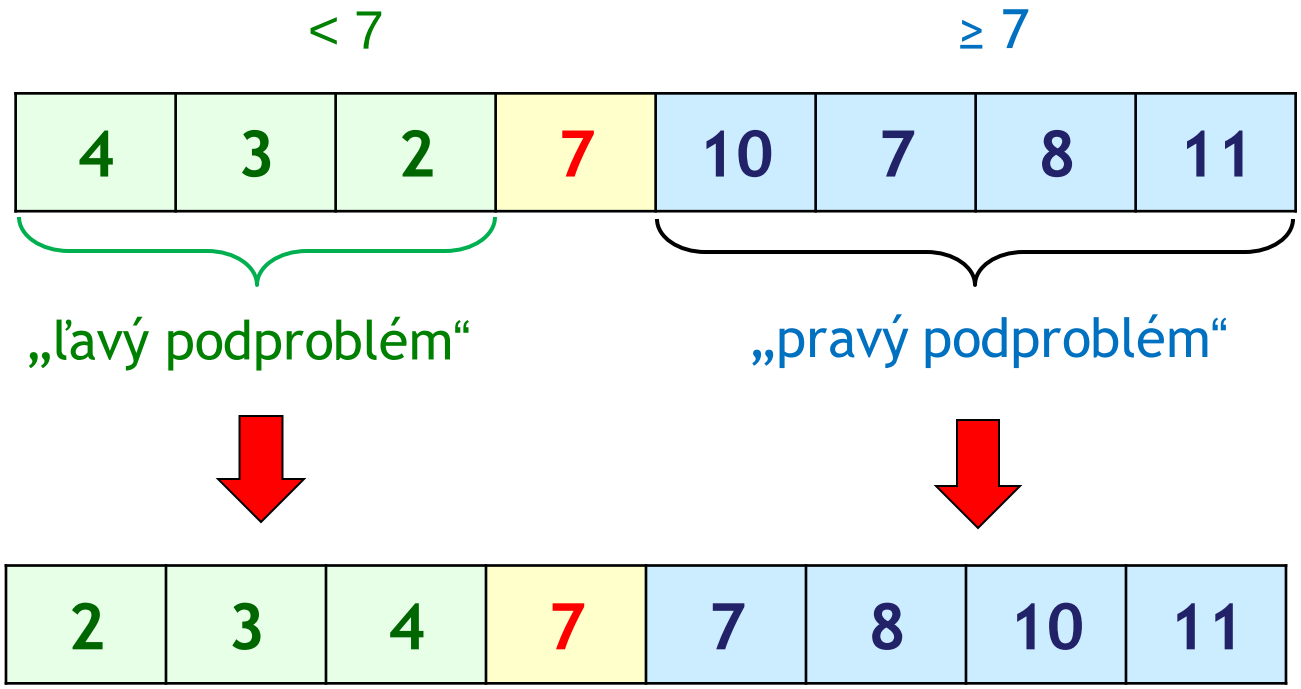
4	11	10	8	3	7	2	7

- Zoberme posledný **prvok** (7) ako tzv. **pivota**...
- Postavme čísla menšie ako pivot **naľavo** od pivota a čísla väčsie alebo rovné ako pivot **napravo**.

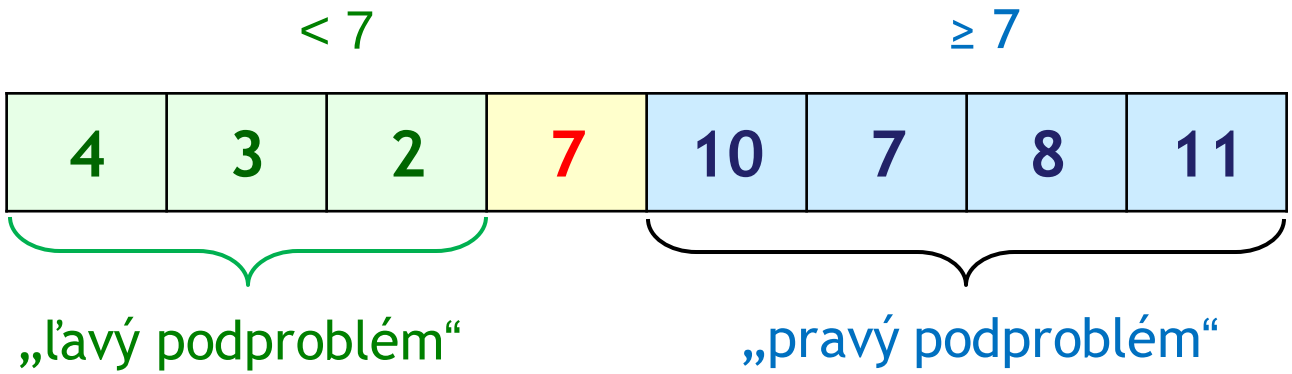
4	3	2	7	10	7	8	11

- Usporiadajme nejako čísla vľavo a čísla vpravo...

Myšlienka č. 1



Myšlienka č. 1



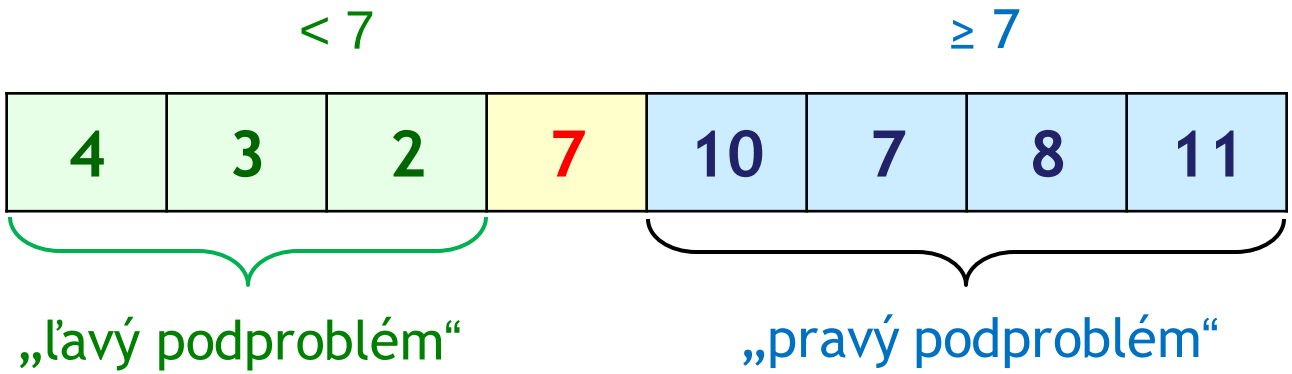

```
public static void utried(int[] p, int odIdx, int poIdx) {
    if (odIdx >= poIdx)
        return;

    int pivotIdx = rozdel(p, odIdx, poIdx);
    utried(p, odIdx, pivotIdx - 1);
    utried(p, pivotIdx + 1, poIdx);
}
```

Preusporiada prvky podpoľa podľa **pivota** a vráti jeho index v podpoli.

Myšlienka č. 1

- aplikujeme rovnaký postup aj na podproblémy

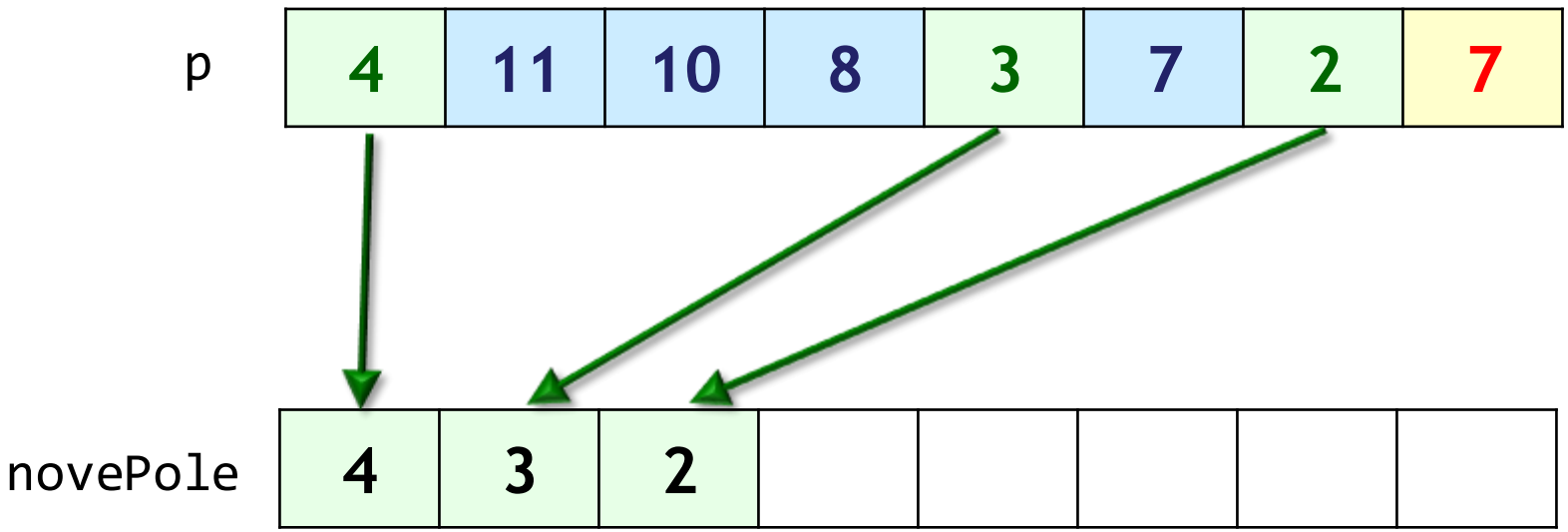
Myšlienka č. 1 - ako na to?



Hlavný problém:

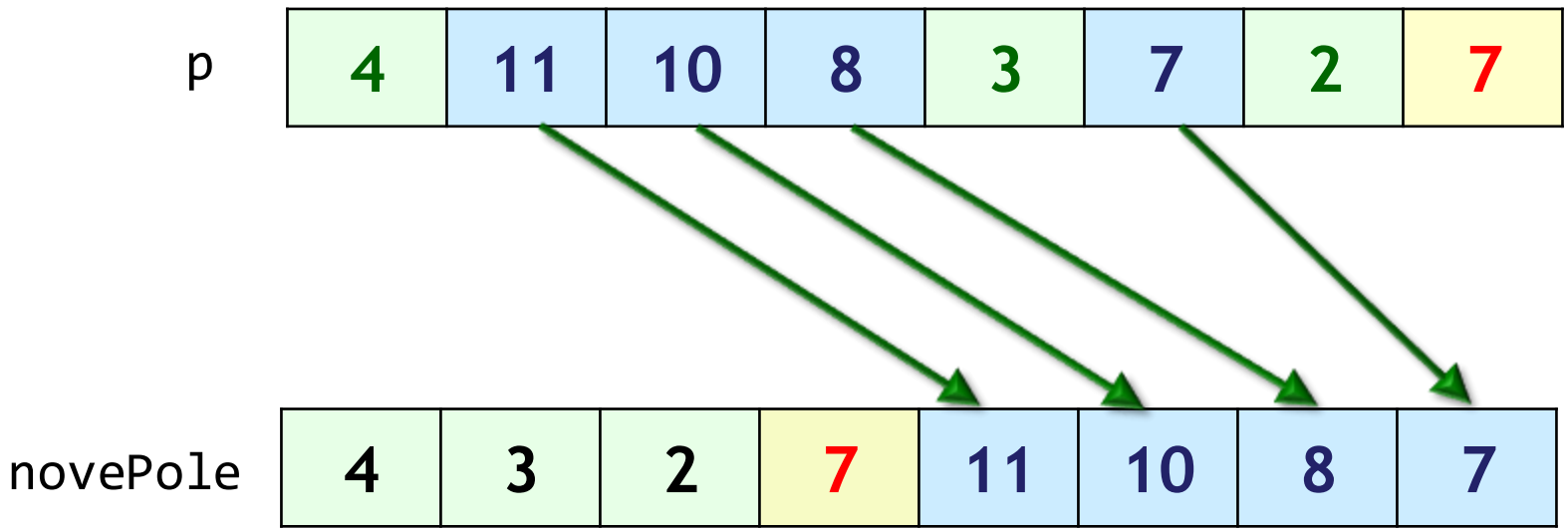
Ako preusporiadať prvky v poli tak, aby prvky naľavo boli menšie a prvky napravo od pivota boli väčšie alebo rovné ako pivot?

Rozdelenie podľa pivota




```
int[] novePole = new int[p.length];
int idx = 0;
for (int i = 0; i < p.length - 1; i++)
    if (p[i] < pivot) {
        novePole[idx] = p[i];
        idx++;
    }
```

Rozdelenie podľa pivota




```
novePole[idx] = pivot;
idx++;

for (int i = 0; i < p.length - 1; i++)
    if (p[i] >= pivot) {
        novePole[idx] = p[i];
        idx++;
    }
```

Rozdelenie podľa pivota

- Celkom 2 prechody poľom: čas $O(n)$ 
- Treba použiť **pomocné** pole... 

Dá sa s tým niečo spraviť?

- **Pozorovanie:**
  - pri „kopírovaní“ ľavej časti platí, že `idx <= i`, t.j. miesto kam zapisujeme (určené `idx`), nikdy **nepredbehne** miesto, ktoré „vyhodnocujeme“ na podmienku, že je menšie ako pivot

Kartičková ukážka...

Rozdelenie bez pomoci

```
public static int rozdel(int p[], int odIdx, int poIdx) {
    int pivot = p[poIdx];

    int idx = odIdx;
    for (int i = odIdx; i <= poIdx - 1; i++)
        if (p[i] < pivot) {
            vymen(p, i, idx);
            idx++;
        }

    vymen(p, idx, poIdx);
    return idx;
}
```

Pivota (posledný prvok) vynechávame

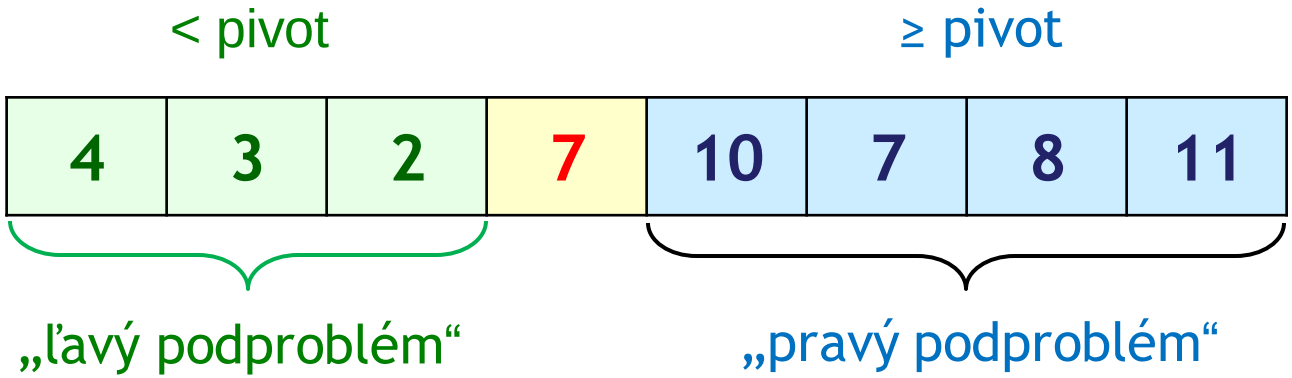
Čas $O(n)$

Namiesto kopírovania vymeníme hodnoty (keďže `idx <= i`)

Pivota z posledného miesta presunieme tam, kam patrí (za prvky menšie od neho)

Všetky prvky väčšie/rovné ako pivot budú až za `idx`. Prečo?

QuickSort



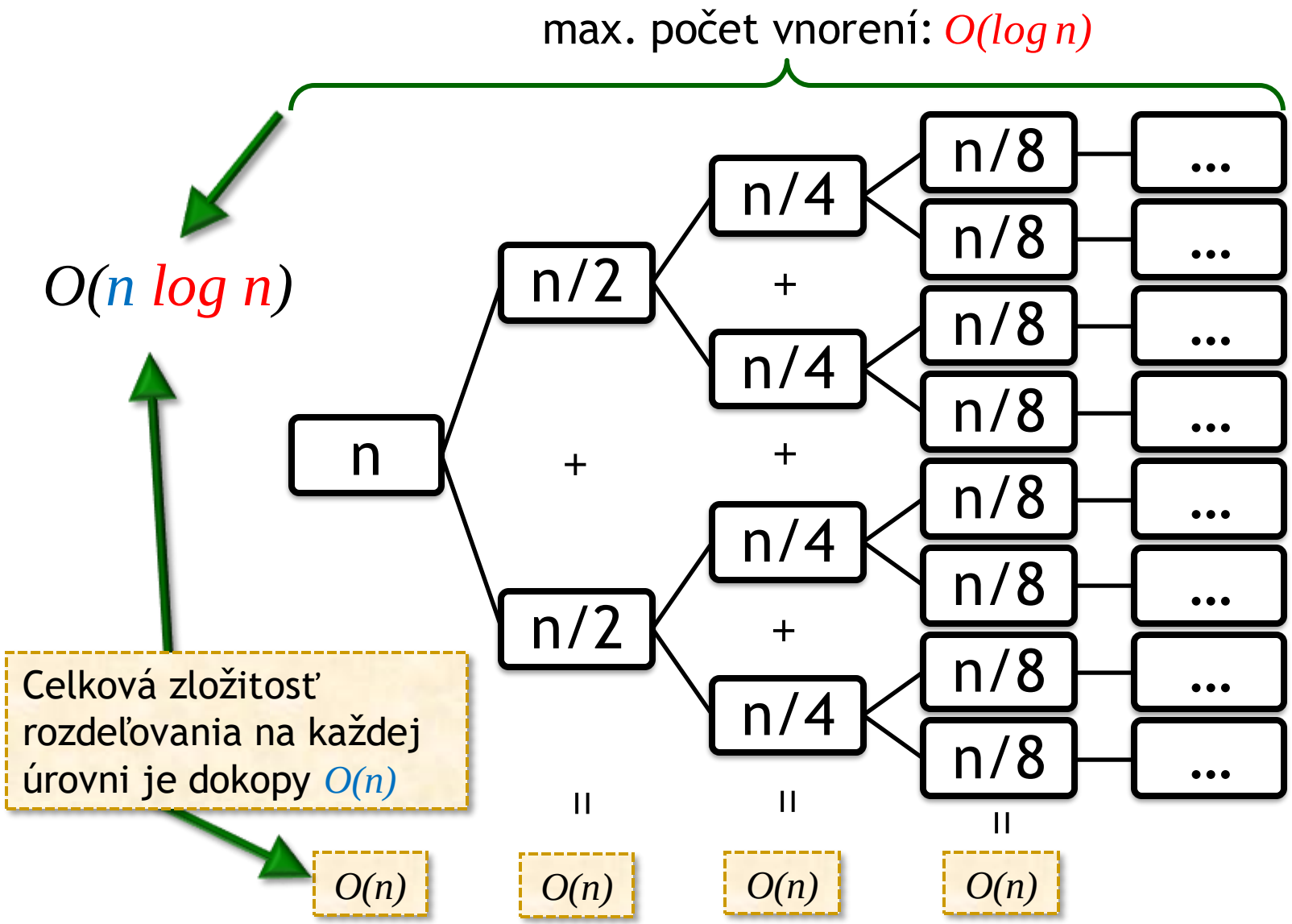

```
static void quickSort(int[] p, int odIdx, int poIdx) {
    if (odIdx >= poIdx)
        return;

    int pivotIdx = rozdel(p, odIdx, poIdx);
    quickSort(p, odIdx, pivotIdx - 1);
    quickSort(p, pivotIdx + 1, poIdx);
}
```

Časová zložitosť rozdelenia n prvkov je $O(n)$

Ako rýchly je QuickSort?

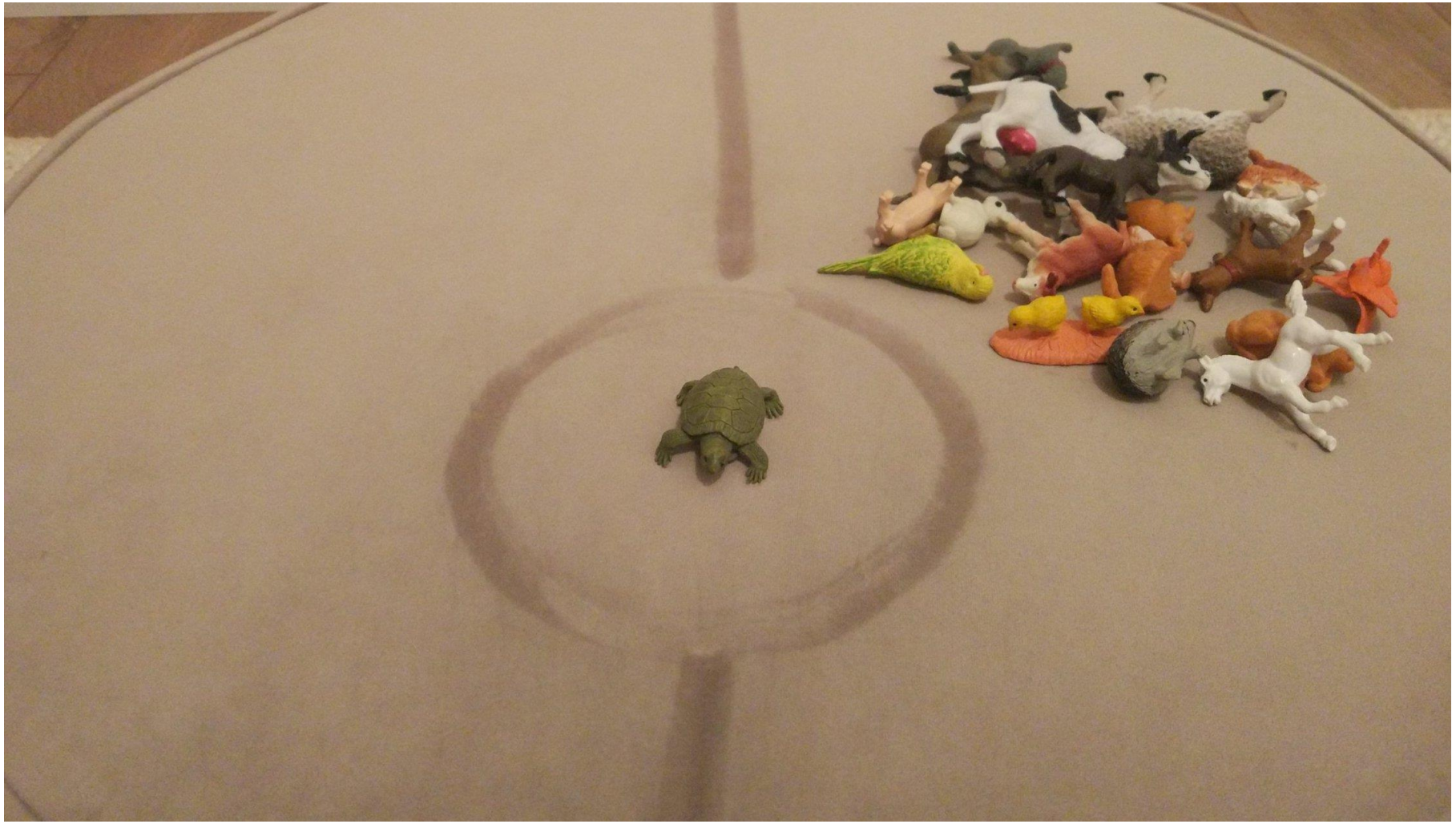
- **Ideálny scenár**: pivot je **mediánom**
  - **medián** je prvok, ktorý je po usporiadaní hodnôt v strede („v polovici“)
  - metóda `rozdel` rozdelí problém (pole) na dva **podobne veľké** podproblémy (podpolia)

QuickSort – ideálny prípad



Výber pivota

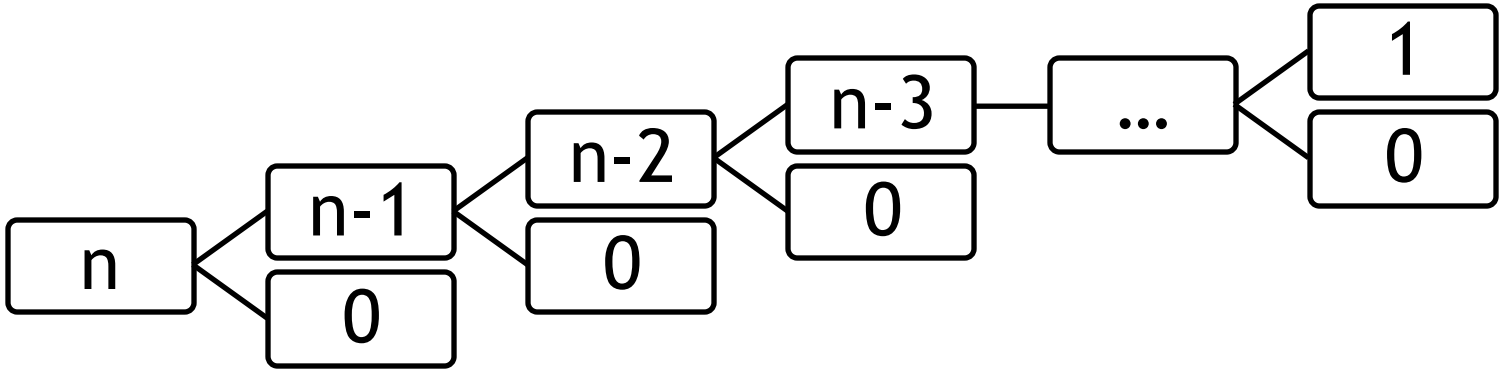
Výber pivota



Výber pivota

QuickSort – najhorší prípad

- **Vstup**: usporiadaná postupnosť
  - ako pivota volíme posledný, t.j. **najväčší prvok**
  - rozdelenie v tvare [n-1:**pivot**:0]



- Časová zložitosť:

$$n + (n\text{-}1) + (n\text{-}2) + \ldots + 1 = \Theta(n^2)$$

QuickSort – sumarizácia

- Rekurzívne triedenie založené na myšlienke **rozdeliť** n-prvkový vstup v čase $O(n)$ na dve skupiny, ktoré sa potom **nezávisle utriedia**:
  - ľavá skupina: **menšie** ako zvolený pivot
  - pravá skupina: **väčšie alebo rovné** ako zvolený pivot
- Netreba žiadnu pomocnú „pamäť“
- Časová zložitosť: „medzi $O(n \log n)$ a $O(n^2)$“
- Teoretická analýza ukázala, že časová zložitosť je v **priemernom prípade**:

$$\boldsymbol{O(n \log n)}$$

QuickSort – a čo ďalej?

Dá sa garantovať čas $O(n \log n)$ pre nejaký triediaci algoritmus vždy?

t.j. nielen v priemernom prípade...

Myšlienka č. 2

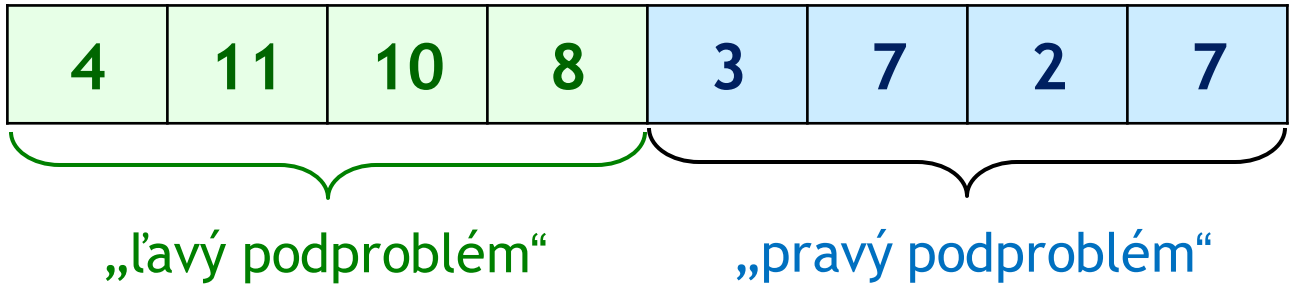
- rozdeľme si zvieratká presne na polovicu

- usporiadajme obe polovice (rovnakým spôsobom) a spojme výsledok dokopy

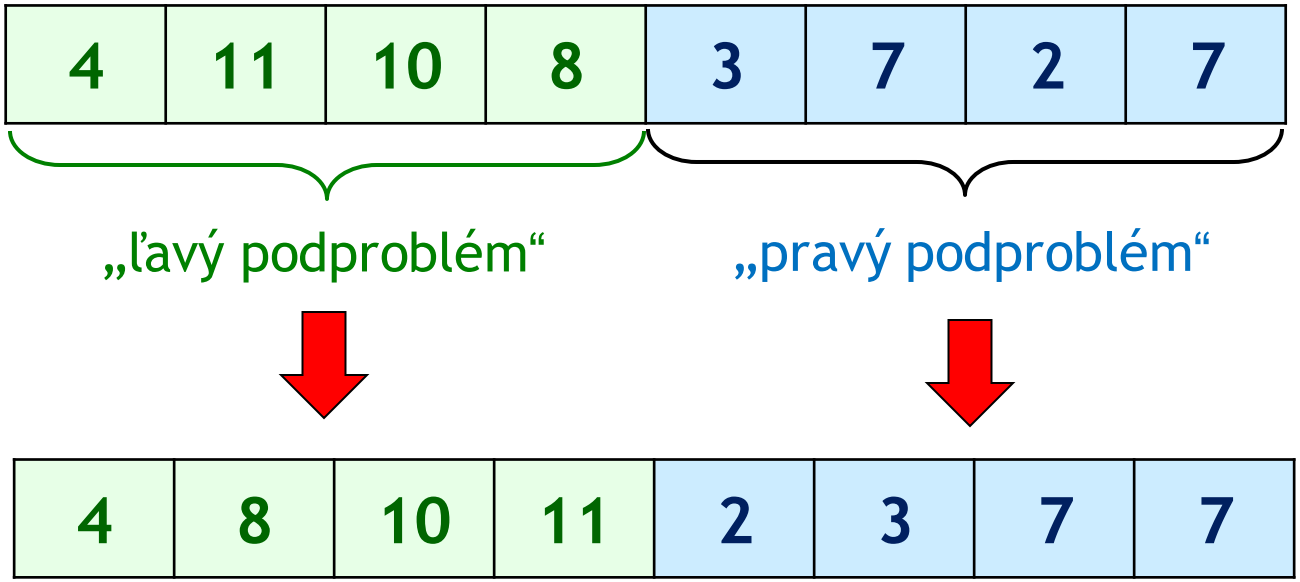
Myšlienka č. 2

- **QuickSort** zlyháva pretože pivotizácia **negarantuje rozdelenie** problému na **rovnako veľké** podproblémy...

- **Riešenie**: rozdeľme vstup „presne“ na polovicu



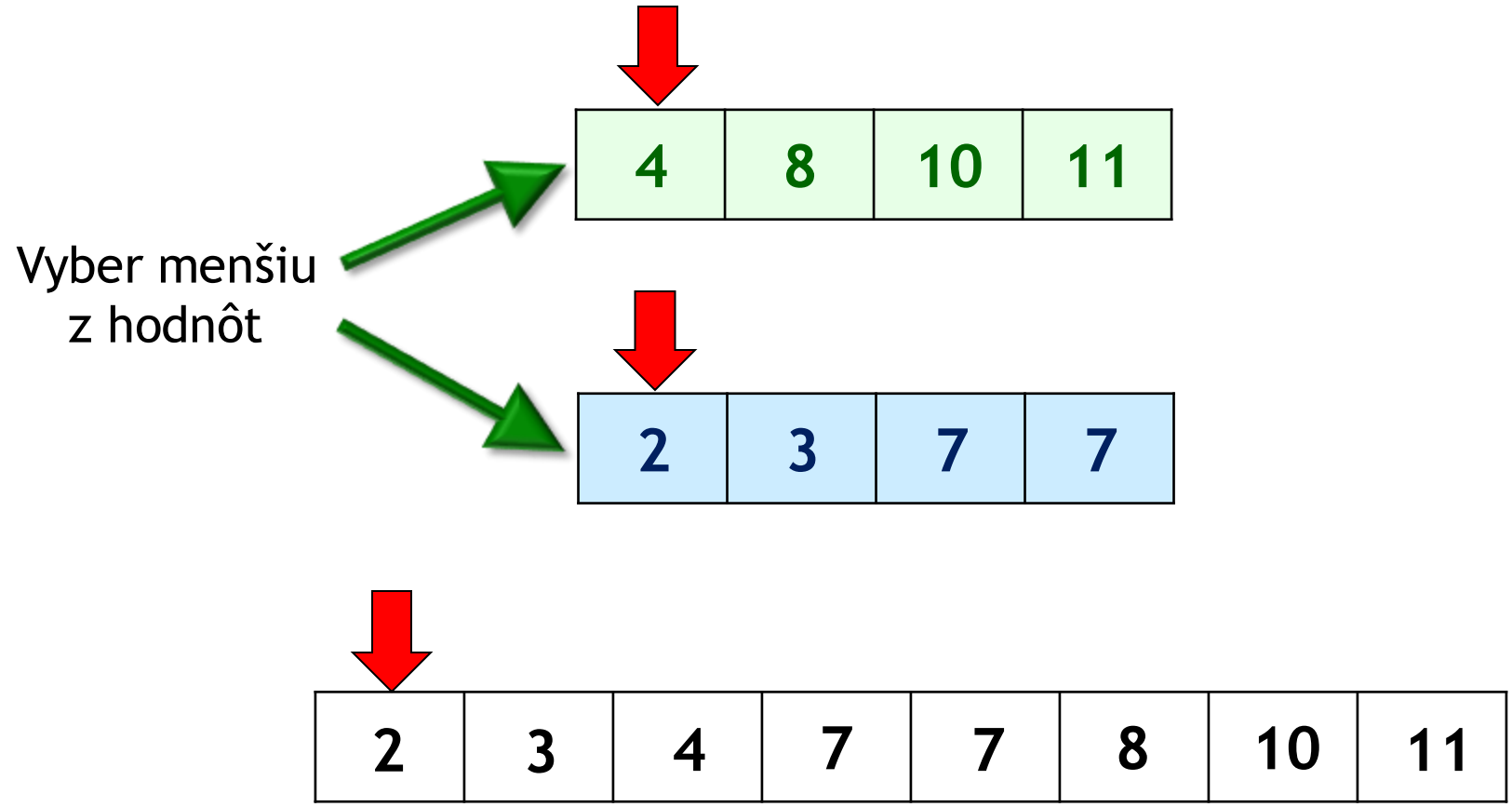
Myšlienka č. 2 (MergeSort)



Ako čo najrýchlejšie „zlepiť“ dve usporiadané postupnosti do jednej usporiadanej postupnosti?

Demonštrácia

Zlučovanie



Časová zložitosť: $O(n)$

MergeSort

4	11	10	8	3	7	2	7

```
public static void mergeSort(int[] p, int odIdx, int poIdx) {
    if (odIdx == poIdx)
        return;

    int stredIdx = (odIdx + poIdx) / 2;
    mergeSort(p, odIdx, stredIdx);
    mergeSort(p, stredIdx + 1, poIdx);



    zluc(p, odIdx, stredIdx, poIdx);
}
```

Utriedime ľavú a pravú časť...

Zlúčime utriedené časti v čase $O(n)$

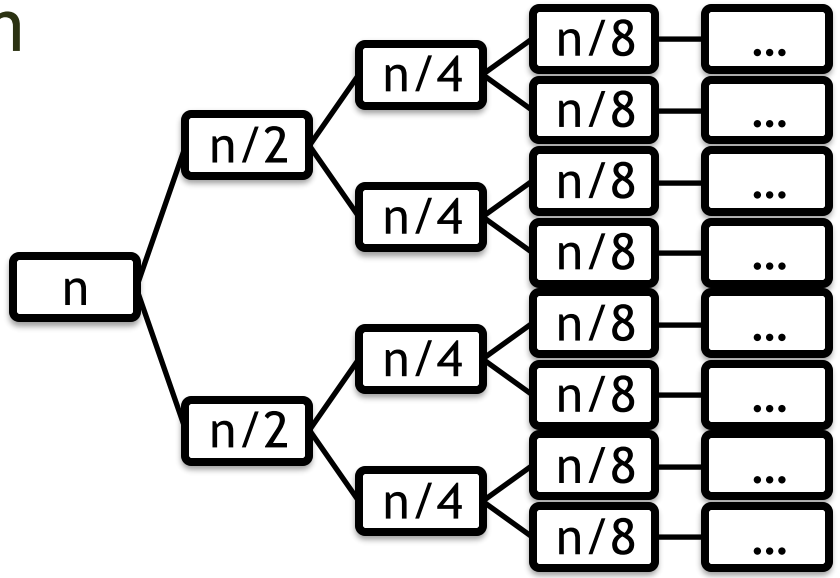
Zlučovanie...

```
public static void zluc(int[] p, int odIdx, int stredIdx, int poIdx) {
  int[] pomocne = new int[poIdx - odIdx + 1];
  int idx1 = odIdx; int poIdx1 = stredIdx;
  int idx2 = stredIdx + 1; int poIdx2 = poIdx;
  int zapisIdx = 0;
  while ((idx1 <= poIdx1) && (idx2 <= poIdx2)) {
    if (p[idx1] <= p[idx2]) {
      pomocne[zapisIdx] = p[idx1]; idx1++; zapisIdx++;
    } else {
      pomocne[zapisIdx] = p[idx2]; idx2++; zapisIdx++;
    }
  }
  while (idx1 <= poIdx1) {
    pomocne[zapisIdx] = p[idx1]; idx1++; zapisIdx++;
  }
  while (idx2 <= poIdx2) {
    pomocne[zapisIdx] = p[idx2]; idx2++; zapisIdx++;
  }
  System.arraycopy(pomocne, 0, p, odIdx, pomocne.length);
}
```

Aky rýchly je MergeSort?

- **Zlučovanie** n prvkov trvá $O(n)$:
  - v každom kroku po $O(1)$ operáciách presunieme jeden prvok postupnosti

- Keďže problém delíme na „polovice“ je celková časová zložitosť v každom behu rovnaká ako v ideálnom prípade **QuickSortu**:
  O(n log n)



MergeSort - sumarizácia

- Rekurzívne triedenie založené na myšlienke rozdeliť postupnosť na **dve rovnako veľké** podpostupnosti, utriediť ich nezávisle a potom v čase $O(n)$ usporiadané podpostupnosti **zlúčiť do usporiadanej** postupnosti
- Čas: $O(n \log n)$ ... vždy
- **Nevýhoda**: treba pomocné pole 
  - vylepšenie: vyrobiť pomocné pole len raz (veľkosti n)
- Dá sa „zbaviť“ pomocného poľa?
  - Áno, ale to už je poriadne iný príbeh ...

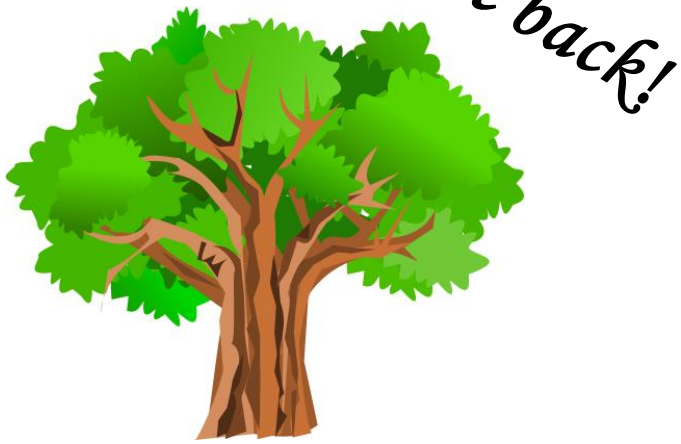
MergeSort – a čo ďalej?

Existuje triediaci algoritmus, ktorý:

- má časovú zložitosť $O(n \log n)$,
- nepotrebuje pomocné pole,
- učí sa na štandardných kurzoch o algoritmoch?

Myšlienka č. 3

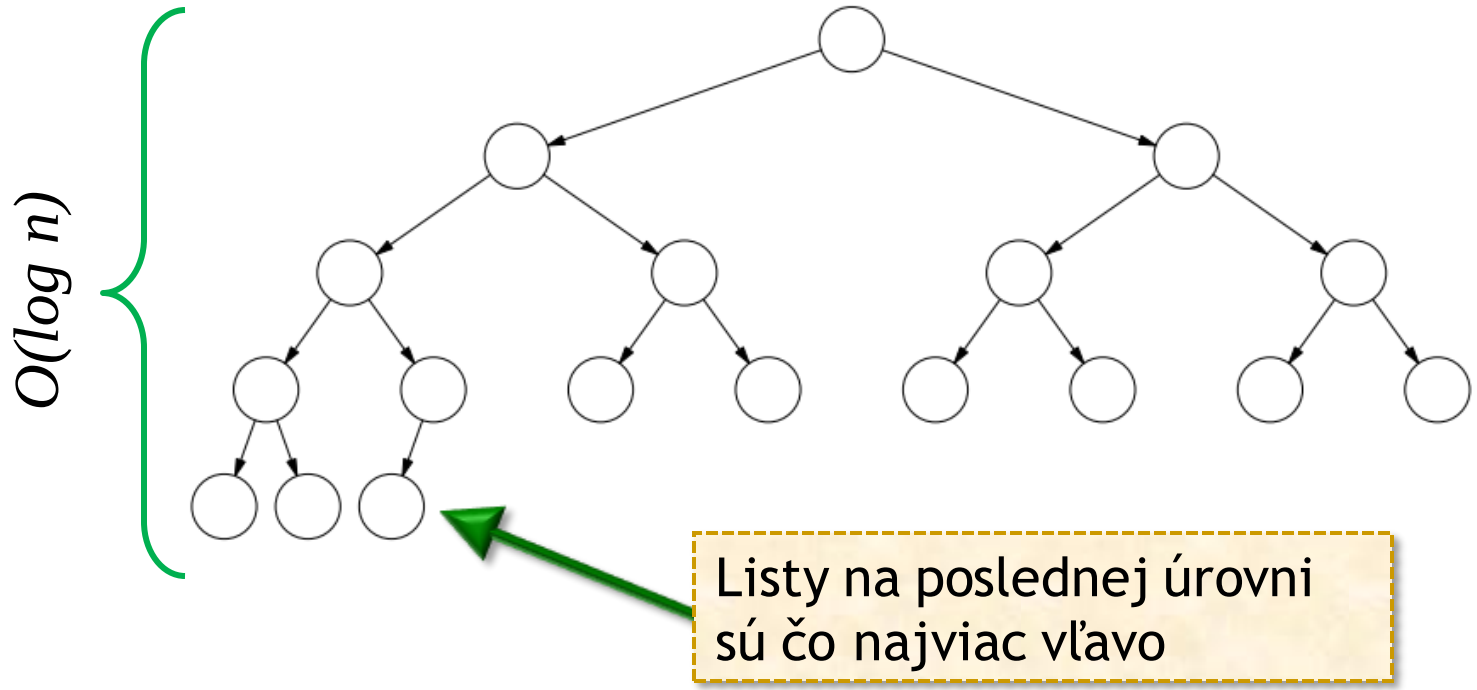
- Zabudnime na triedenie...



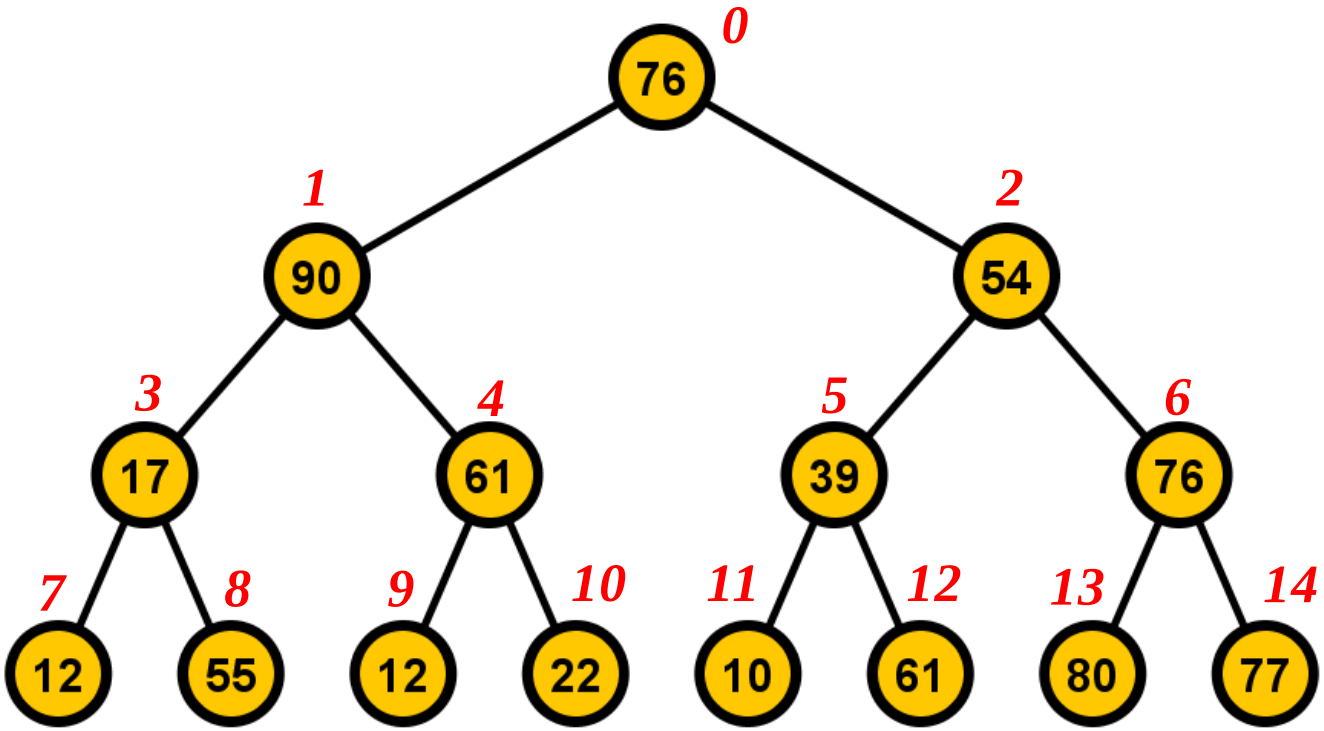
- **Binárna halda** (binary heap) je:
  - **takmer úplný binárny strom**
  - pre každý uzol (s výnimkou koreňa) platí, že jeho **rodič** obsahuje **väčšiu** alebo **rovnakú** hodnotu

Takmer úplný binárny strom

- T.Ú.B.S. je binárny strom, v ktorom:
  - všetky úrovne (s výnimkou poslednej) **musia byť plné**
  - na poslednej úrovni sú všetky listy „čo najviac vľavo“

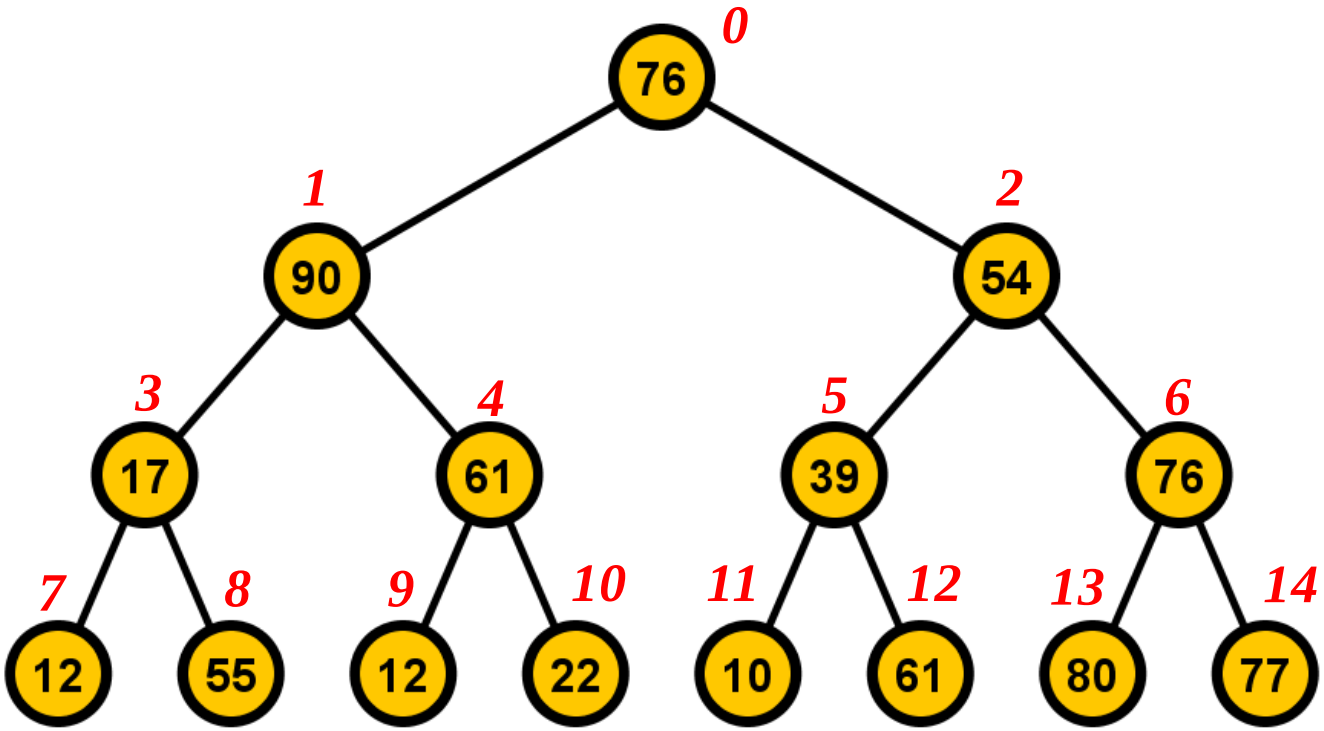


Takmer úplny bin. strom v poli



0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
76	90	54	17	61	39	76	12	55	12	22	10	61	80	77

Takmer úplny bin. strom v poli

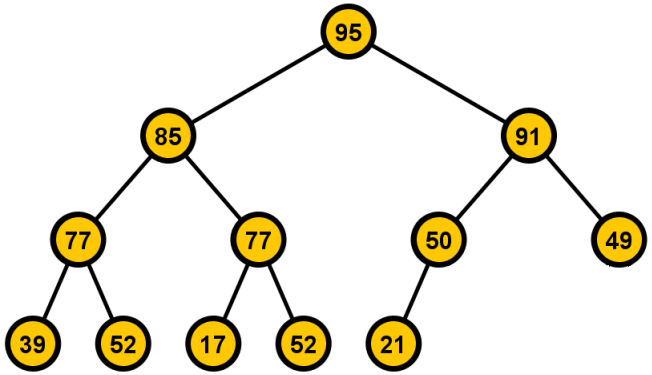


Uzol na indexe ***i***, má:

- ľavého syna na indexe ***2i+1***
- pravého syna na indexe ***2i+2***

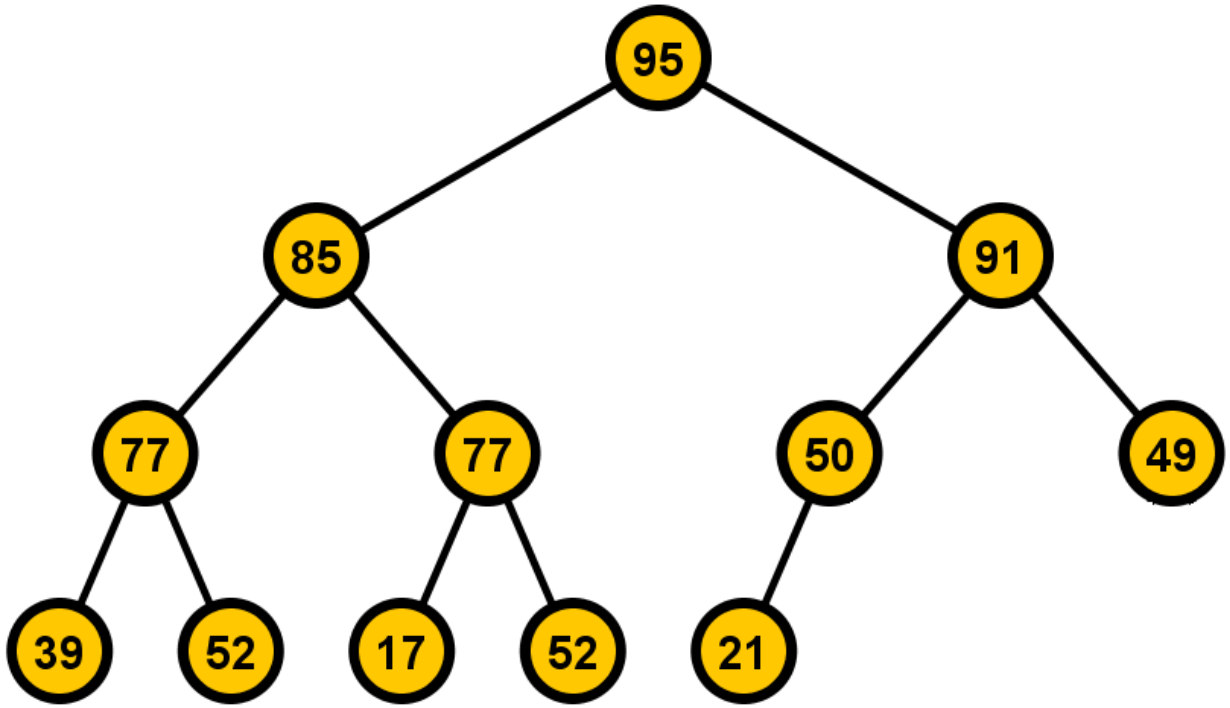
Takmer úplné binárne stromy

- **Vlastnosti:**
  - logaritmická výška
  - ide ich „efektívne“ uložiť v poli
  - na každé n-prvkové pole môžeme pozerať ako na nejaký takmer úplný binárny strom, ktorý je uložený v tomto poli



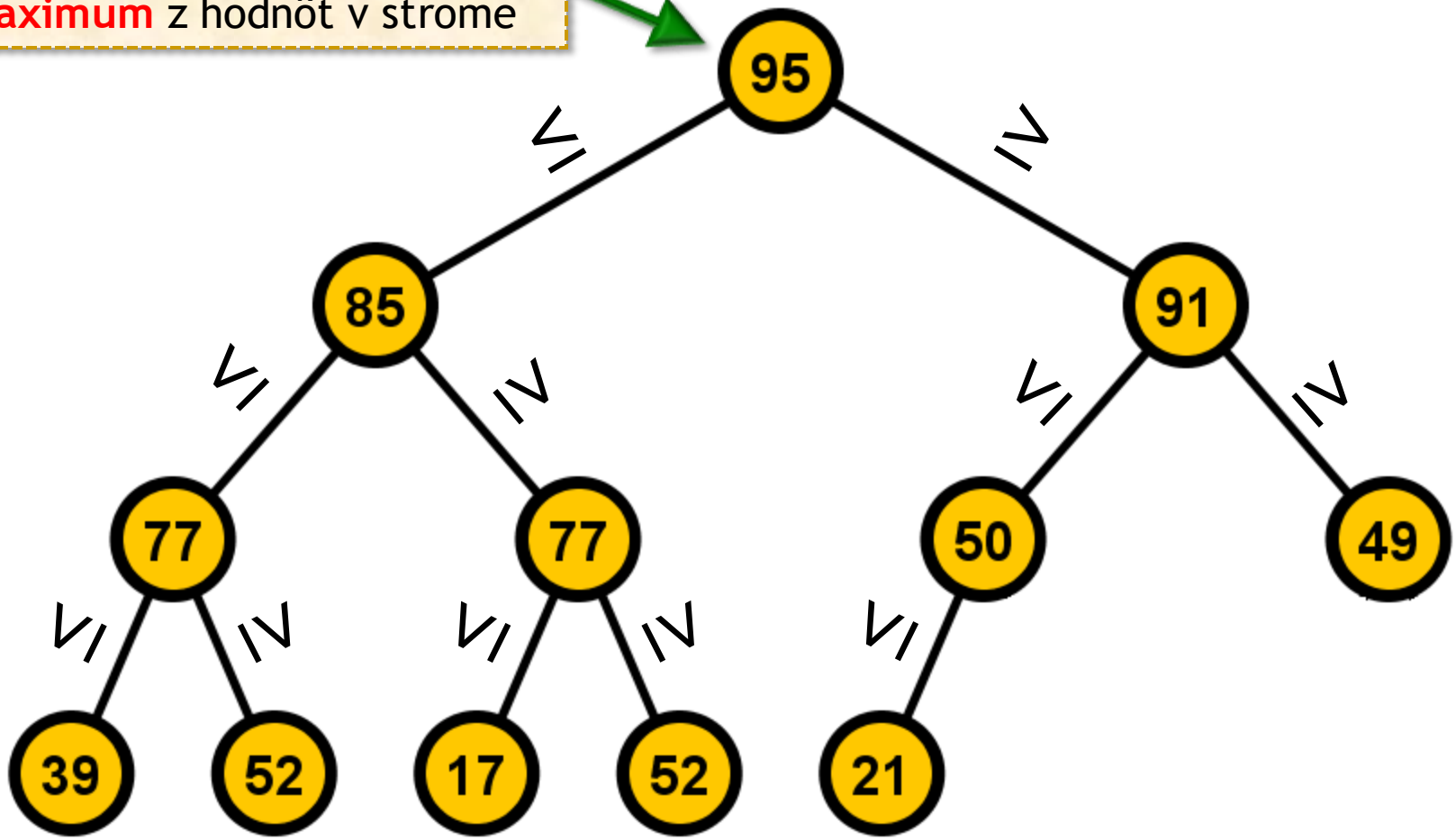
Halda (Heap)

- **Binárna halda** je:
  - takmer úplný binárny strom
  - pre každý uzol (s výnimkou koreňa) platí, že jeho rodič obsahuje väčšiu alebo rovnakú hodnotu



Vlastnosti haldy

V koreni (pod)stromu je vždy **maximum** z hodnôt v strome



Halda zvierat

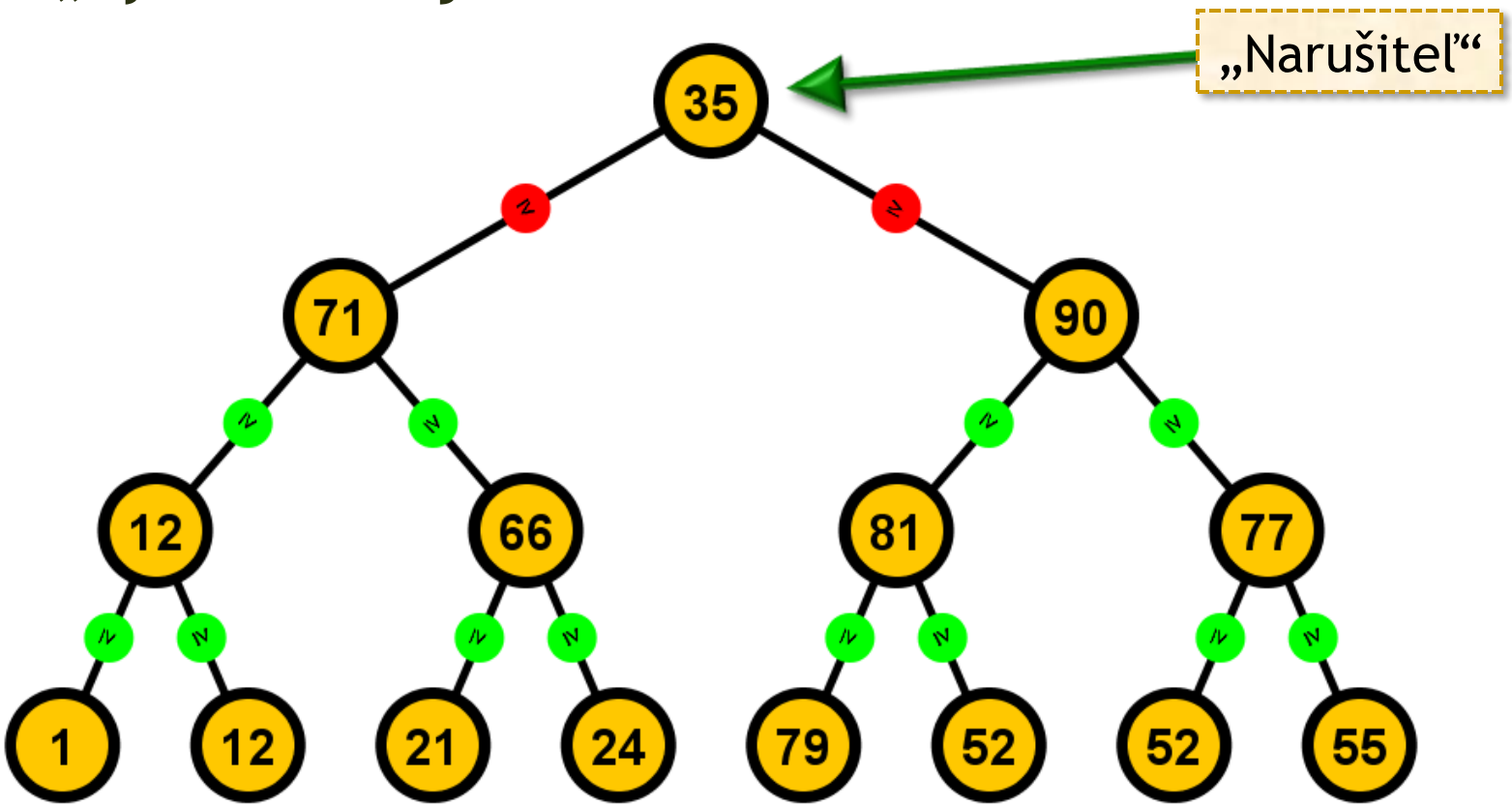
Ako sa vyrábajú haldy?

Ako ľubovoľný takmer úplný binárny strom prerobiť na haldu obsahujúcu tie isté hodnoty?

alebo problém hromadnej fotografie...

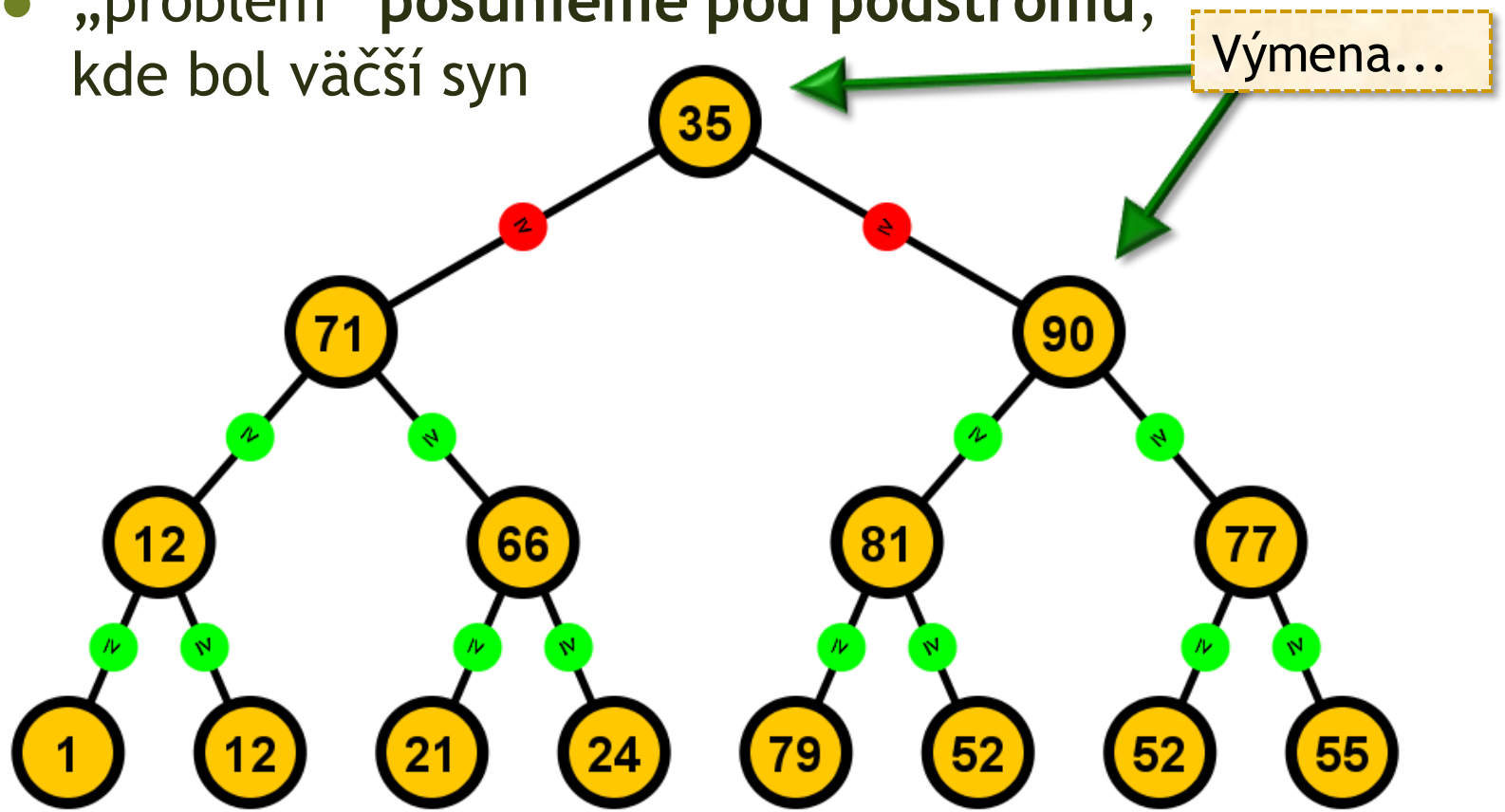
Ako strom prerobiť na haldu?

- Riešime jednoduchší problém:
  - strom je skoro halda – jediný „**narušiteľ**“ vlastnosti „byť haldou“ je koreň...



Ako strom prerobiť na haldu?

- Vymeňme narušiteľa **s väčším z jeho synov**:
  - v koreni bude najväčšia hodnota (to chceme)
  - „problém“ **posunieme pod podstromu**, kde bol väčší syn

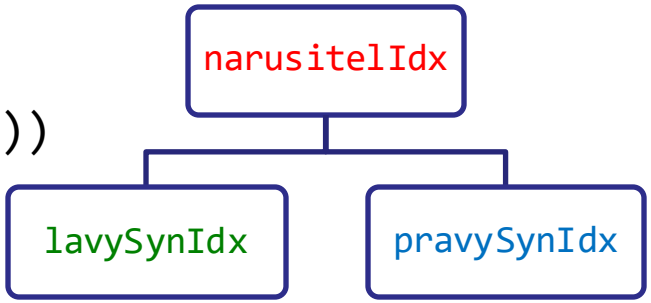


Haldovací algoritmus

```
void uhalduj(int[] p, int narusitelIdx, int poslednyIdx) {
  while (true) {
    int najvacsiIdx = narusitelIdx;
    int lavySynIdx = narusitelIdx * 2 + 1;
    int pravySynIdx = narusitelIdx * 2 + 2;

    if ((lavySynIdx <= poslednyIdx) &&
            (p[lavySynIdx] > p[najvacsiIdx]))
      najvacsiIdx = lavySynIdx;

    if ((pravySynIdx <= poslednyIdx) &&
            (p[pravySynIdx] > p[najvacsiIdx]))
      najvacsiIdx = pravySynIdx;

    if (najvacsiIdx == narusitelIdx)
      break;

    vymen(p, narusitelIdx, najvacsiIdx);
    narusitelIdx = najvacsiIdx;
  }
}
```

II
p.length-1

Kto z nich je najväčší?



Ak je potenciálny „narušiteľ“ nie menší ako jeho synovia, môžeme skončiť...

„Narušiteľa“ vymeníme s väčším zo synov...

Haldovací algoritmus

- Časová zložitosť:
  - každý krok (iterácia `while` cyklu) je $O(1)$
  - každým krokom „narušiteľa“ **posunieme o úroveň nižšie**
  - „narušiteľ“ prestane byť „narušiteľom“ najneskôr vtedy, keď sa stane listom
  - n-prvková halda má nanajvýš $O(\log n)$ úrovní

$$O(\log n)$$

Ako prerobiť strom na haldu?

- **Fakty:**
  - každý podstrom haldy je halda... (rekurzia)
  - v logaritmickom čase vieme opraviť haldu s „narušiteľom“ haldovosti v koreni
- **Riešenie:**
  - budujeme haldu **zdola-nahor**
  - po tom, čo sú ľavý aj pravý podstrom (rekurzívne) prerobené na haldy, jediný možný „narušiteľ“ je v koreni...

Demonštrácia

Haldujeme celý strom!

Vylepšenie: `p.length / 2`

```
void vytvorHaldu(int[] p) {
    for (int korenIdx = p.length - 1; korenIdx >= 0; korenIdx--)
        uhalduj(p, korenIdx, p.length - 1);
}
```

Postupne od najmenších podstromov haldujeme celý strom (všetky podstromy s koreňom väčším ako `korenIdx` sú už halda)

- Časová zložitosť:
  - n-krát uhaldovanie v $O(\log n) = O(n \log n)$
  - detailnejšou analýzou (na cvičeniach?) možno ukázať, že zložitosť celého haldovania je dokonca ***O(n)***
    - „haldujeme od koreňa“ veľa stromov s malou výškou a málo stromov s veľkou výškou...

Od haldovania k triedeniu

- **Fakty o halde:**
  - v **koreni** je vždy **maximum** z hodnôt
- **Triediaci algoritmus (idea):**
  - **vyber koreň haldy** (maximum z hodnôt) a zober ho ako **posledný prvok** usporiadanej postupnosti
  - do koreňa haldy daj prvok haldy **s najväčším indexom** a zmenši haldu o 1
    - ak tento prvok naruší vlastnosť byť haldou, tak je to „narušiteľ“ v koreni – toto vieme riešiť v logaritmickom čase...
  - usporiadaj zvyšné prvky v halde rovnakým spôsobom

HeapSort

```
public static void heapSort(int[] p) {
    vytvorHaldu(p);

    for (int i = p.length - 1; i > 0; i--) {
        vymen(p, 0, i);
        uhalduj(p, 0, i - 1);
    }
}
```

Vytvoríme z hodnôt v poli haldu: $O(n)$

Vymeníme koreň s „posledným“ prvkom haldy: $O(1)$

Výmena možno narušila haldu v koreni, preto ju skúsime opraviť: $O(\log n)$

Veľkosť haldy po každom kroku zmenšujeme

Celkový čas:

$$O(n) + n \,.O(\log n) = O(n) + O(n \log n) = \mathbf{O(n \log n)}$$

HeapSort – sumarizácia

- Triedenie využívajúce údajovú štruktúru **binárna halda**, ktorú možno efektívne reprezentovať v poli
- Celkový čas: $O(n \log n)$ 
- Netreba pomocné pole 

- Halda a jej modifikácie sa využívajú aj pri iných algoritmoch a údajových štruktúrach...
  - napr. implementácia prioritného radu

Sumarizácia

- `QuickSort` - priemerný čas $O(n \log n)$
  - najhorší prípad $O(n^2)$, existuje verzia garantujúca $O(n \log n)$
  - najčastejšie používaný v praxi
  - T. Hoare, 1960
- `MergeSort` – čas $O(n \log n)$, treba pomocné pole
  - princíp sa používa v databázach pri triedeniach na disku
  - J. von Neumann, 1945
- `HeapSort` – čas $O(n \log n)$
  - J.W.J. Williams, 1964
- Problém triedenia porovnávaním je $\Omega(n \log n)$ - $O(n \log n)$ **nie je možné asymptoticky zlepšiť...**

Ako som usporiadal zvieratká

- malé, stredné a veľké
- 20 zvieratiek = malá veľkosť vstupu

- **asymptotická zložitosť** (... $n>n_0$ ...)
  - malé vstupy na demonštráciu algoritmov

Čo sa skrýva za Arrays.sort

- *(z dokumentácie Javy)*
  Implementation note:
  The sorting algorithm is a **Dual-Pivot Quicksort** by Vladimir Yaroslavskiy, Jon Bentley, and Joshua Bloch.

  This algorithm offers O(n log(n)) performance on many data sets that cause other quicksorts to degrade to quadratic performance, and is **typically faster** than traditional (one-pivot) Quicksort implementations.

Quicksort vs. Heapsort

- **Prečo sa používa Quicksort ak má Heapsort zaručený čas *O(n log n)*?**
- Počet výmen:
  - Quicksort – nerobí zbytočné výmeny
  - Heapsort – vymieňa aj v usporiadanom poli
- **Randomized** Quicksort – nie je závislý od distribúcie prvkov na vstupe
- Quicksort worst case je zriedkavé
- Mergesort, Heapsort majú svoje aplikácie

PAZ

ak nie sú otázky...

Ďakujem za pozornosť!