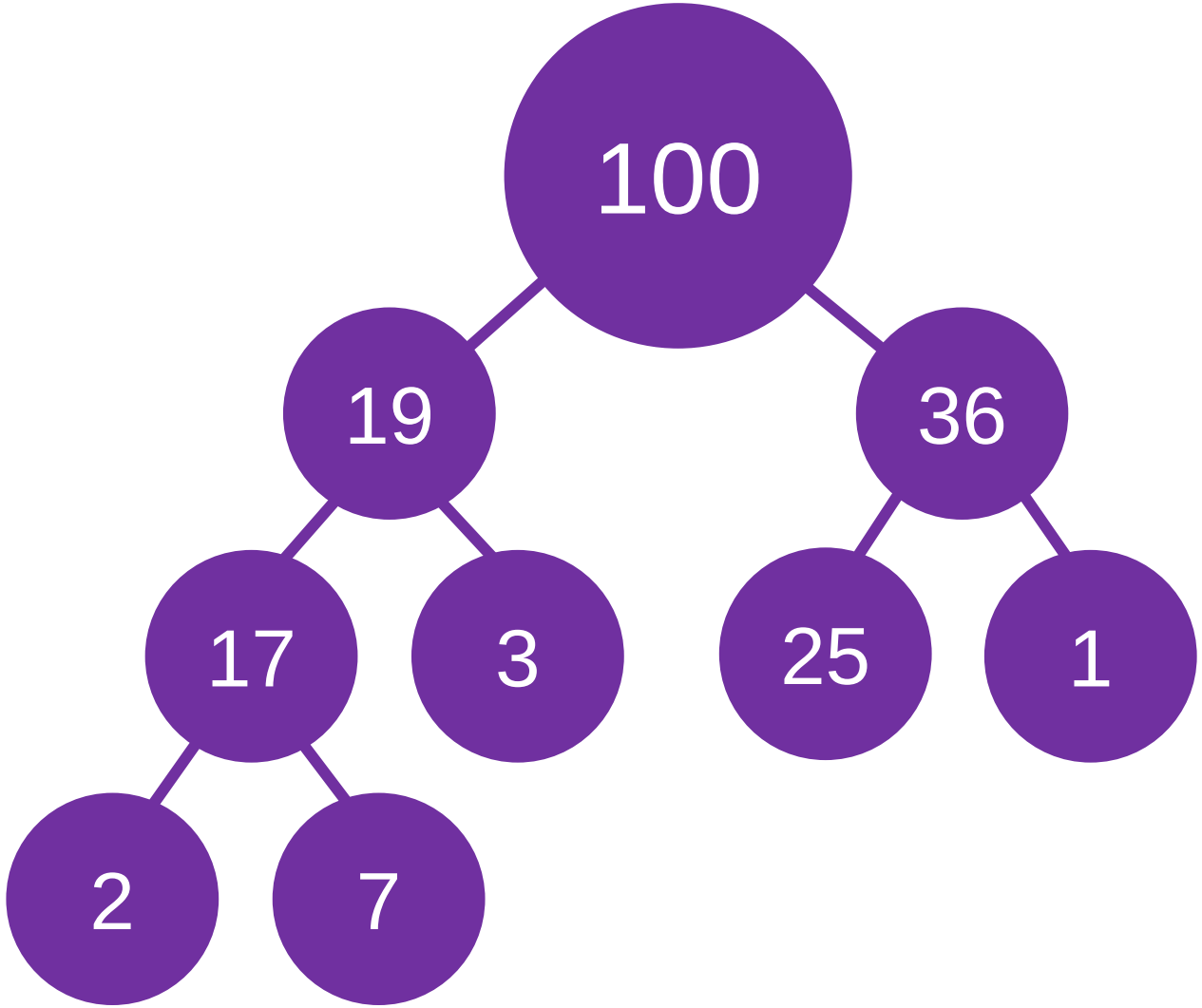
Heap sort

Pavol Marák

Programovacie techniky



OBSAH

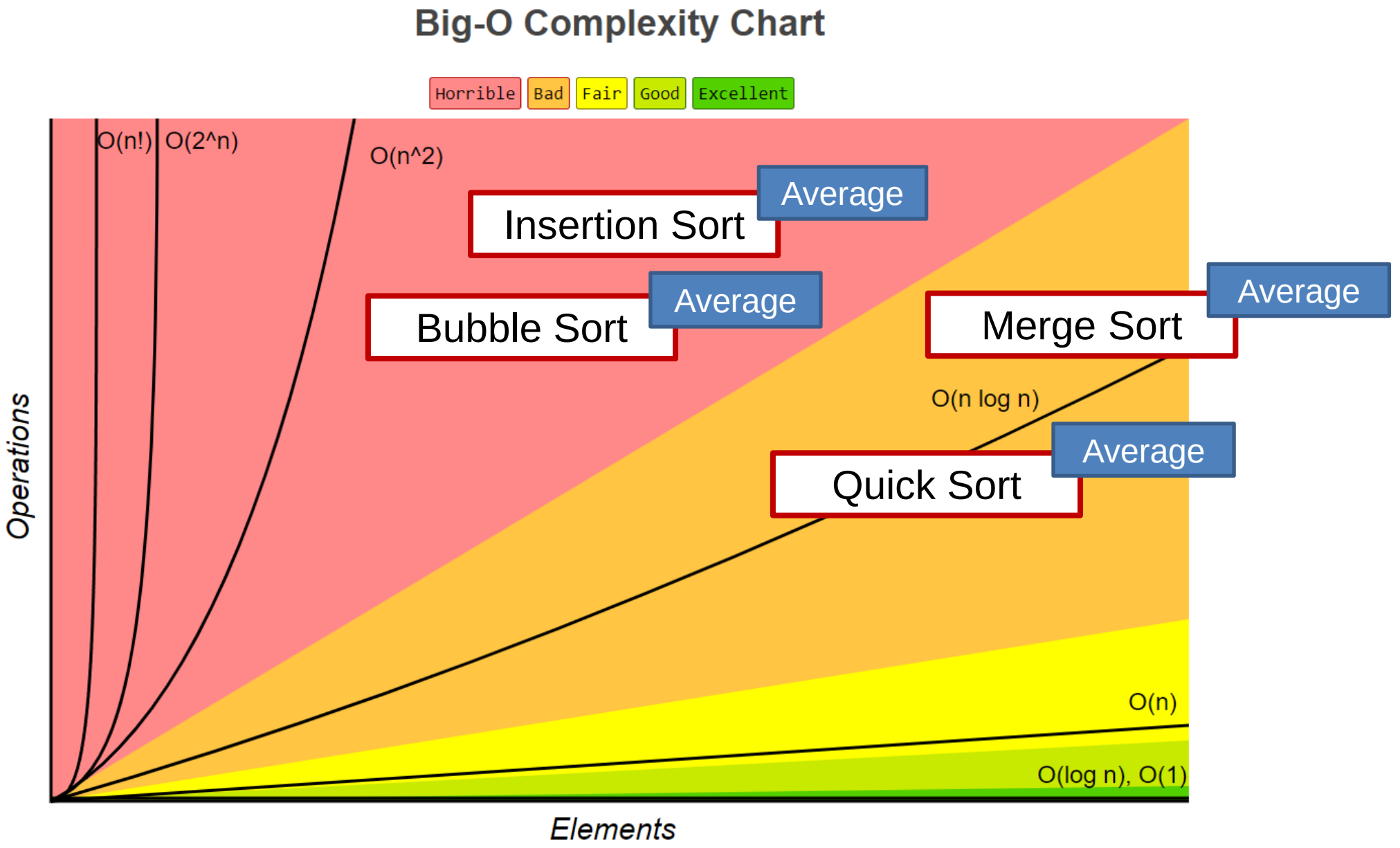
Heap sort

- Vlastnosti
- Algoritmus
- Dátová štruktúra heap
- Reprezentácia binárneho stromu pomocou poľa
- Vizualizácia algoritmu
- Vzorová implementácia v C/C++

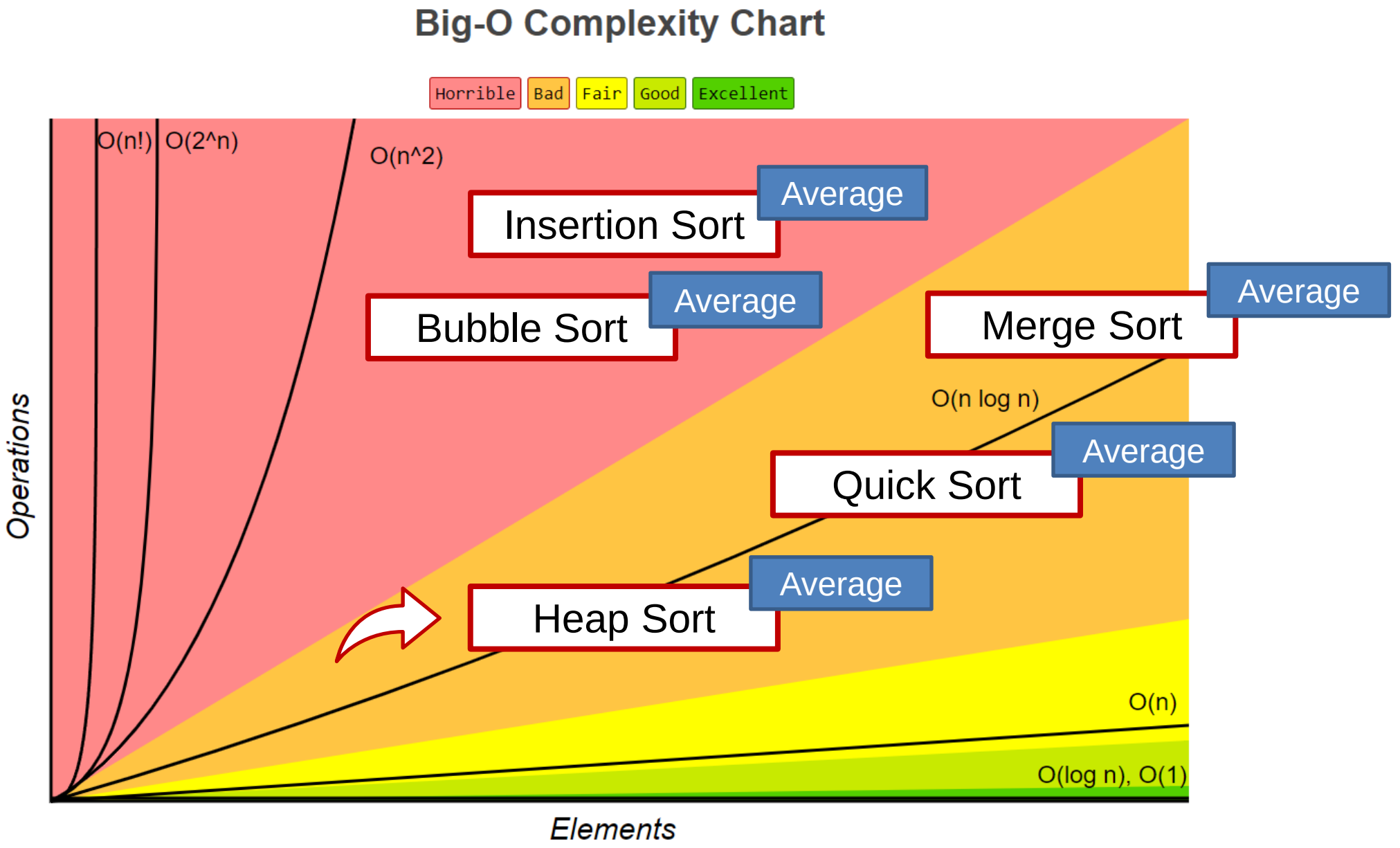
Vlastnosti

Vlastnosti

- Priemerná aj najhoršia zložitosť O(n log n).



Zdroj: https://www.bigocheatsheet.com/



Zdroj: https://www.bigocheatsheet.com/

Array Sorting Algorithms

Algorithm	Time Complexity			Space Complexity
	Best	Average	Worst	Worst
Quicksort	Ω(n log(n))	Θ(n log(n))	O(n^2)	O(log(n))
Mergesort	Ω(n log(n))	Θ(n log(n))	O(n log(n))	O(n)
Timsort	Ω(n)	Θ(n log(n))	O(n log(n))	O(n)
Heapsort	Ω(n log(n))	Θ(n log(n))	O(n log(n))	O(1)
Bubble Sort	Ω(n)	Θ(n^2)	O(n^2)	O(1)
Insertion Sort	Ω(n)	Θ(n^2)	O(n^2)	O(1)
Selection Sort	Ω(n^2)	Θ(n^2)	O(n^2)	O(1)
Tree Sort	Ω(n log(n))	Θ(n log(n))	O(n^2)	O(n)
Shell Sort	Ω(n log(n))	Θ(n(log(n))^2)	O(n(log(n))^2)	O(1)
Bucket Sort	Ω(n+k)	Θ(n+k)	O(n^2)	O(n)
Radix Sort	Ω(nk)	Θ(nk)	O(nk)	O(n+k)
Counting Sort	Ω(n+k)	Θ(n+k)	O(n+k)	O(k)
Cubesort	Ω(n)	Θ(n log(n))	O(n log(n))	O(n)

Zdroj: https://www.bigocheatsheet.com/

Array Sorting Algorithms

Dnes

	Algorithm	Time Complexity			Space Complexity
		Best	Average	Worst	Worst
✓	Quicksort	Ω(n log(n))	Θ(n log(n))	O(n^2)	O(log(n))
✓	Mergesort	Ω(n log(n))	Θ(n log(n))	O(n log(n))	O(n)
	Timsort	Ω(n)	Θ(n log(n))	O(n log(n))	O(n)
✓	Heapsort	Ω(n log(n))	Θ(n log(n))	O(n log(n))	O(1)
✓	Bubble Sort	Ω(n)	Θ(n^2)	O(n^2)	O(1)
✓	Insertion Sort	Ω(n)	Θ(n^2)	O(n^2)	O(1)
	Selection Sort	Ω(n^2)	Θ(n^2)	O(n^2)	O(1)
	Tree Sort	Ω(n log(n))	Θ(n log(n))	O(n^2)	O(n)
	Shell Sort	Ω(n log(n))	Θ(n(log(n))^2)	O(n(log(n))^2)	O(1)
	Bucket Sort	Ω(n+k)	Θ(n+k)	O(n^2)	O(n)
	Radix Sort	Ω(nk)	Θ(nk)	O(nk)	O(n+k)
	Counting Sort	Ω(n+k)	Θ(n+k)	O(n+k)	O(k)
	Cubesort	Ω(n)	Θ(n log(n))	O(n log(n))	O(n)

Zdroj: https://www.bigocheatsheet.com/

Vlastnosti

- Priemerná aj najhoršia zložitosť O(n log n).
- Rozdeľuje vstupné pole na zotriedenú a nezotriedenú časť.

Vlastnosti

- Priemerná aj najhoršia zložitosť O(n log n).
- Rozdeľuje vstupné pole na zotriedenú a nezotriedenú časť.
- Pri triedení sa používa dátová štruktúra **heap.**

Vlastnosti

- Priemerná aj najhoršia zložitosť O(n log n).
- Rozdeľuje vstupné pole na zotriedenú a nezotriedenú časť.
- Pri triedení sa používa dátová štruktúra **heap.**
- "In-place" algoritmus.

Vlastnosti

- Priemerná aj najhoršia zložitosť O(n log n).
- Rozdeľuje vstupné pole na zotriedenú a nezotriedenú časť.
- Pri triedení sa používa dátová štruktúra **heap.**
- "In-place" algoritmus.
- Nie je stabilný.

Algoritmus

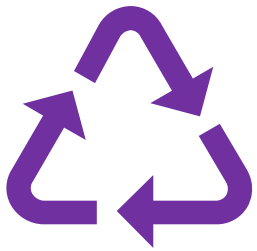
Algoritmus

1. **Fáza**
   - Vytvorenie heap-u z prvkov poľa.

Algoritmus

1. **Fáza**
   - Vytvorenie heap-u z prvkov poľa.
2. **Fáza**

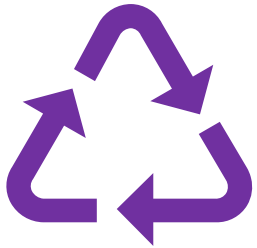
   Opakujeme pokiaľ heap obsahuje prvky:



Algoritmus

1. **Fáza**
   - Vytvorenie heap-u z prvkov poľa.
2. **Fáza**

   Opakujeme pokiaľ heap obsahuje prvky:

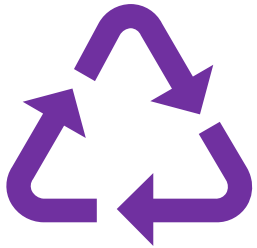


   - Extrakcia maxima/minima z heap-u.

Algoritmus

1. **Fáza**
   - Vytvorenie heap-u z prvkov poľa.
2. **Fáza**

   Opakujeme pokiaľ heap obsahuje prvky:



   - Extrakcia maxima/minima z heap-u.
   - Rekonštrukcia "poškodeného" heap-u.

Heap

Heap

- Má formu binárneho stromu.

Heap

- Má formu binárneho stromu.
- Platí:

Heap

- Má formu binárneho stromu.
- Platí:
    - Kompletný, doľava zarovnaný binárny strom

Heap

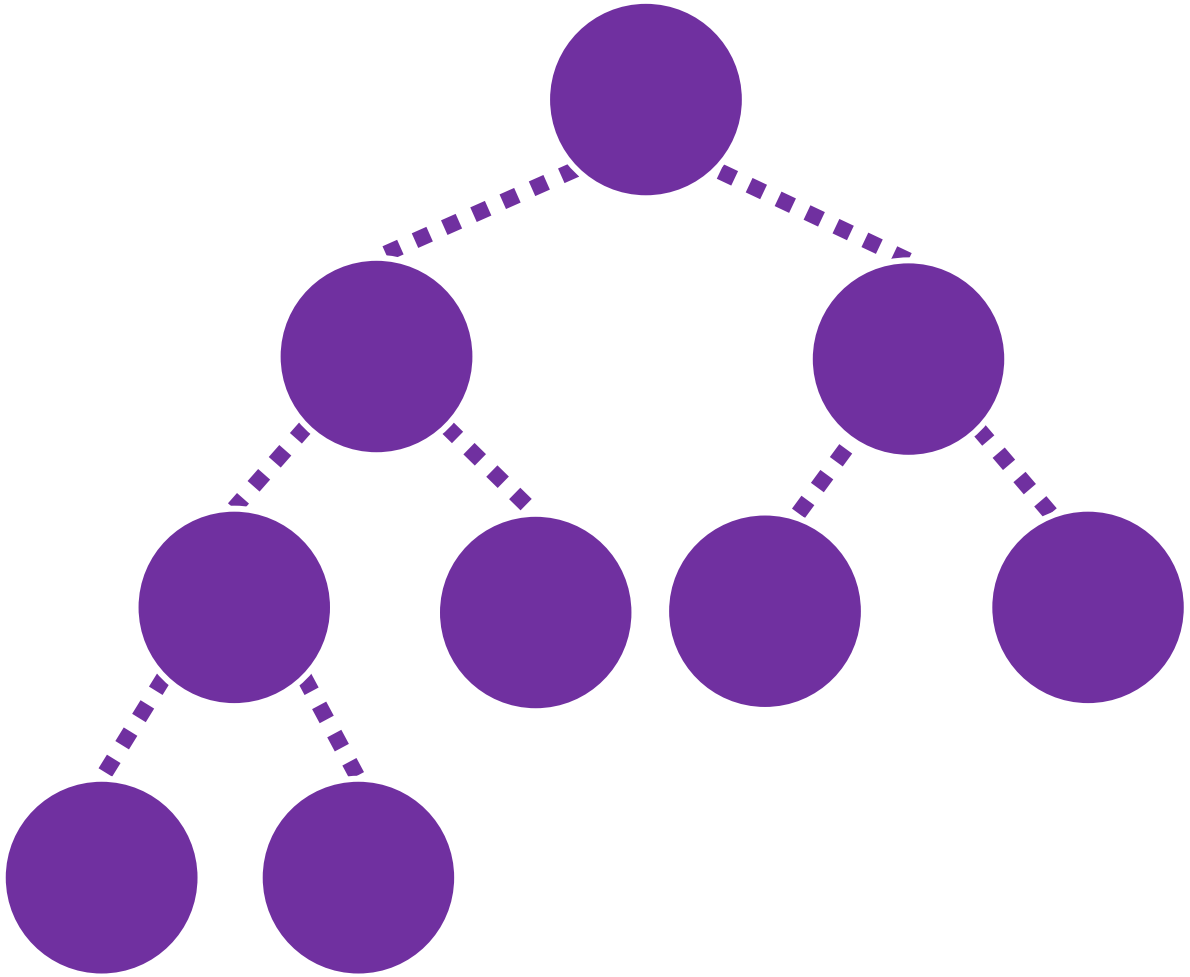
- Má formu binárneho stromu.
- Platí:
  - Kompletný, doľava zarovnaný binárny strom
  - Koreňom stromu je maximum/minimum

Heap

- Má formu binárneho stromu.
- Platí:
  - Kompletný, doľava zarovnaný binárny strom
  - Koreňom stromu je maximum/minimum
  - **Max-heap/Min-heap** vlastnosť

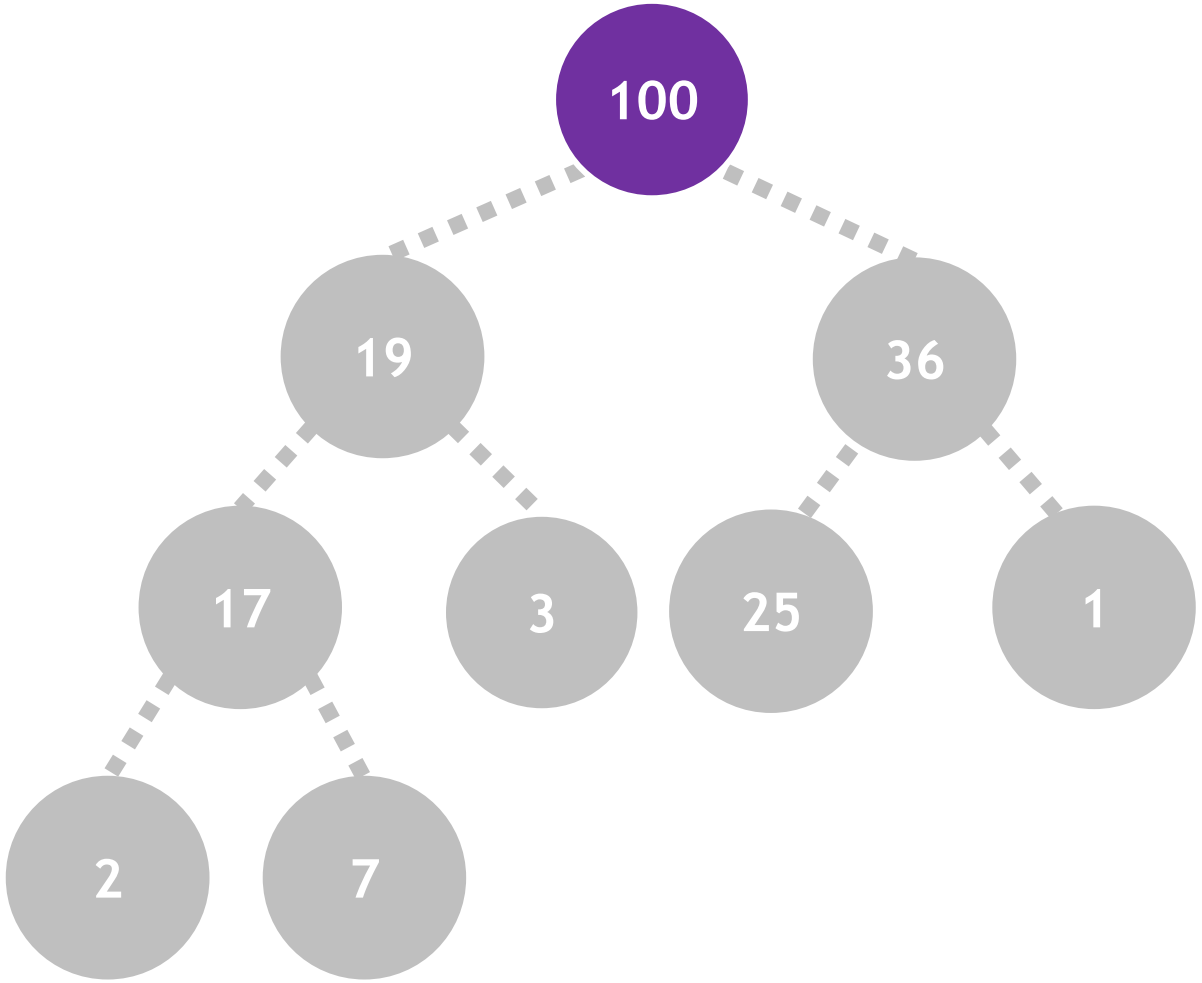
Max-Heap

Kompletný, doľava zarovnaný binárny strom



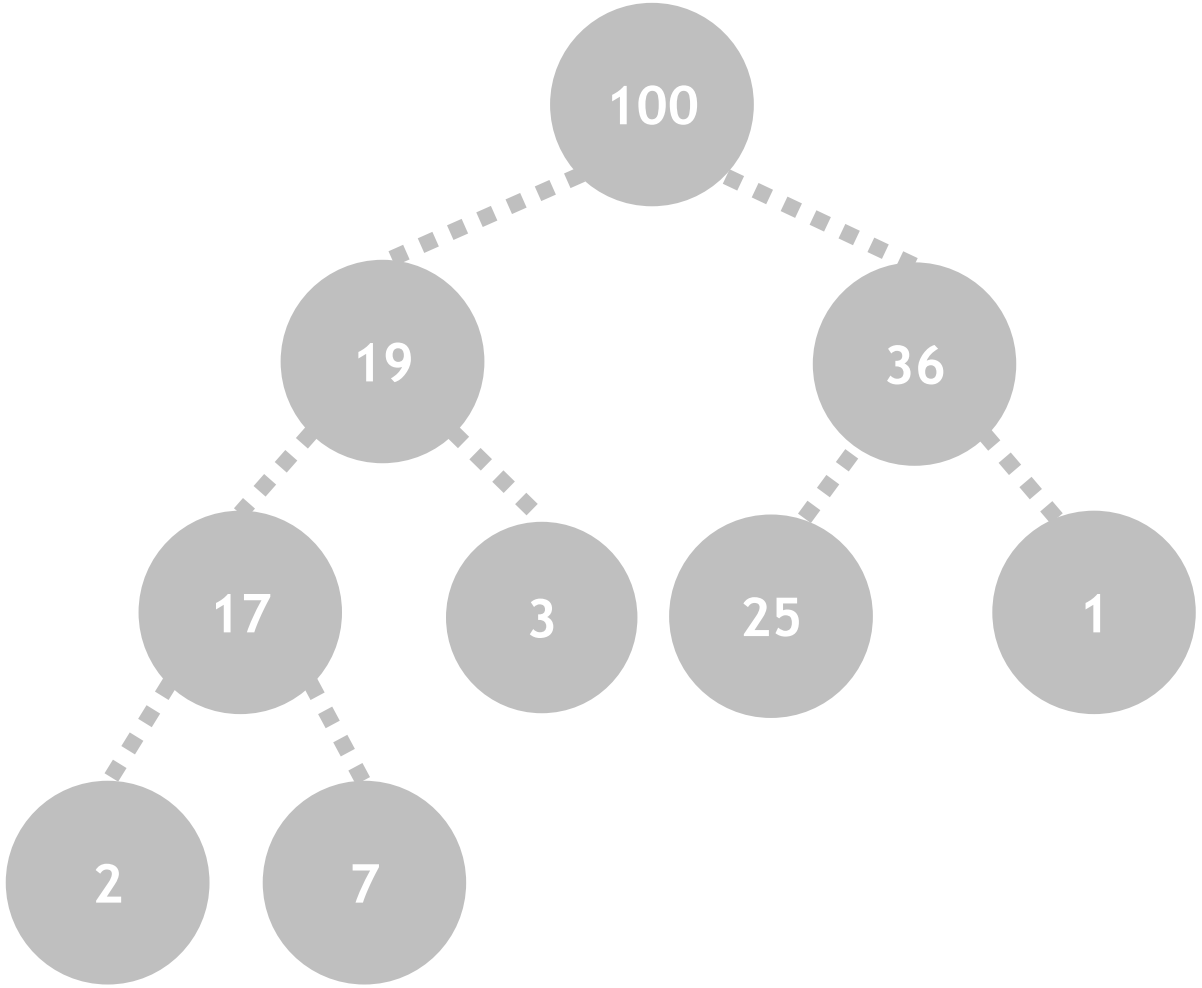
Max-Heap

Koreňom stromu je maximum zo všetkých vrcholov



Max-Heap

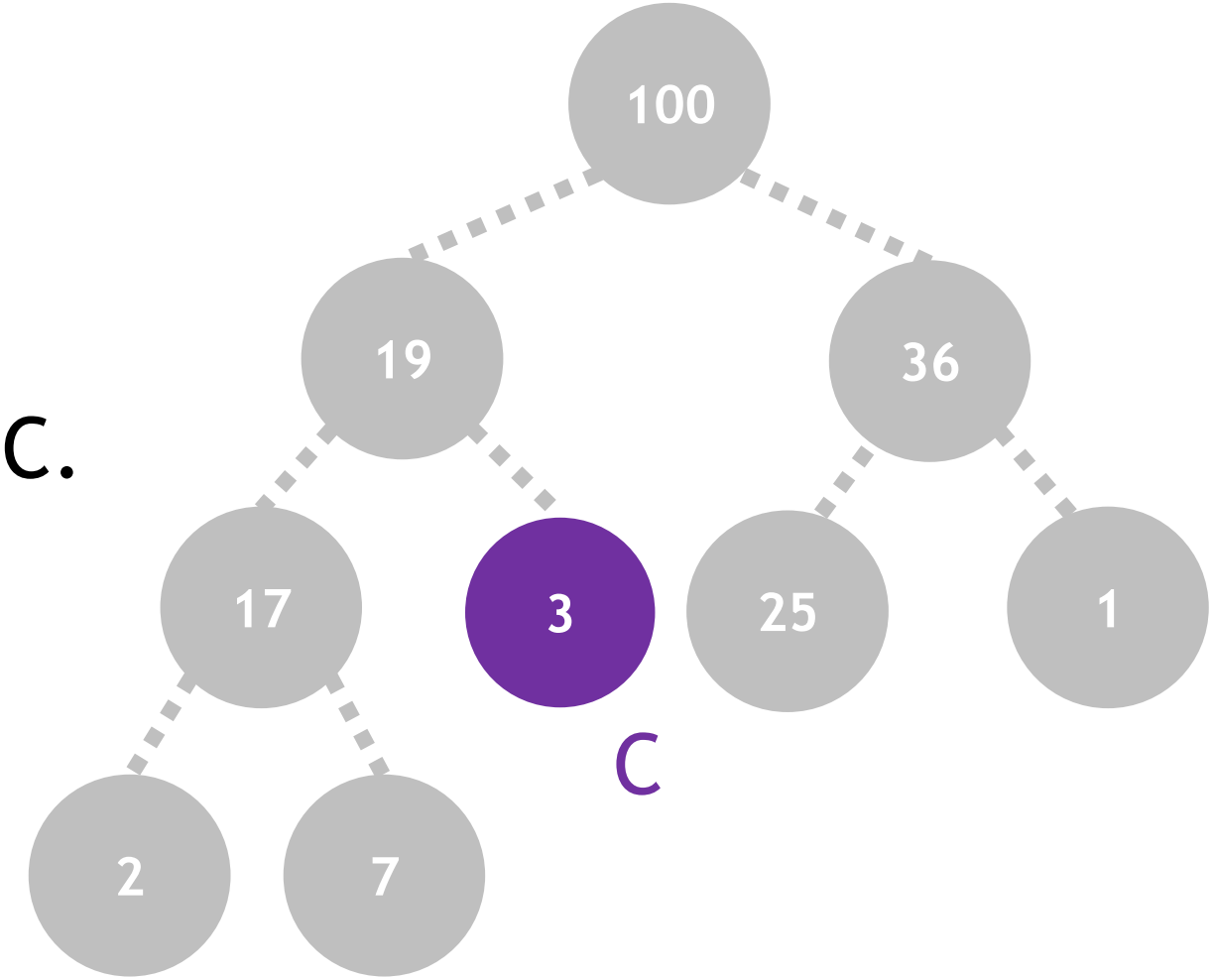
Vlastnosť Max-heap



Max-Heap

Vlastnosť Max-heap

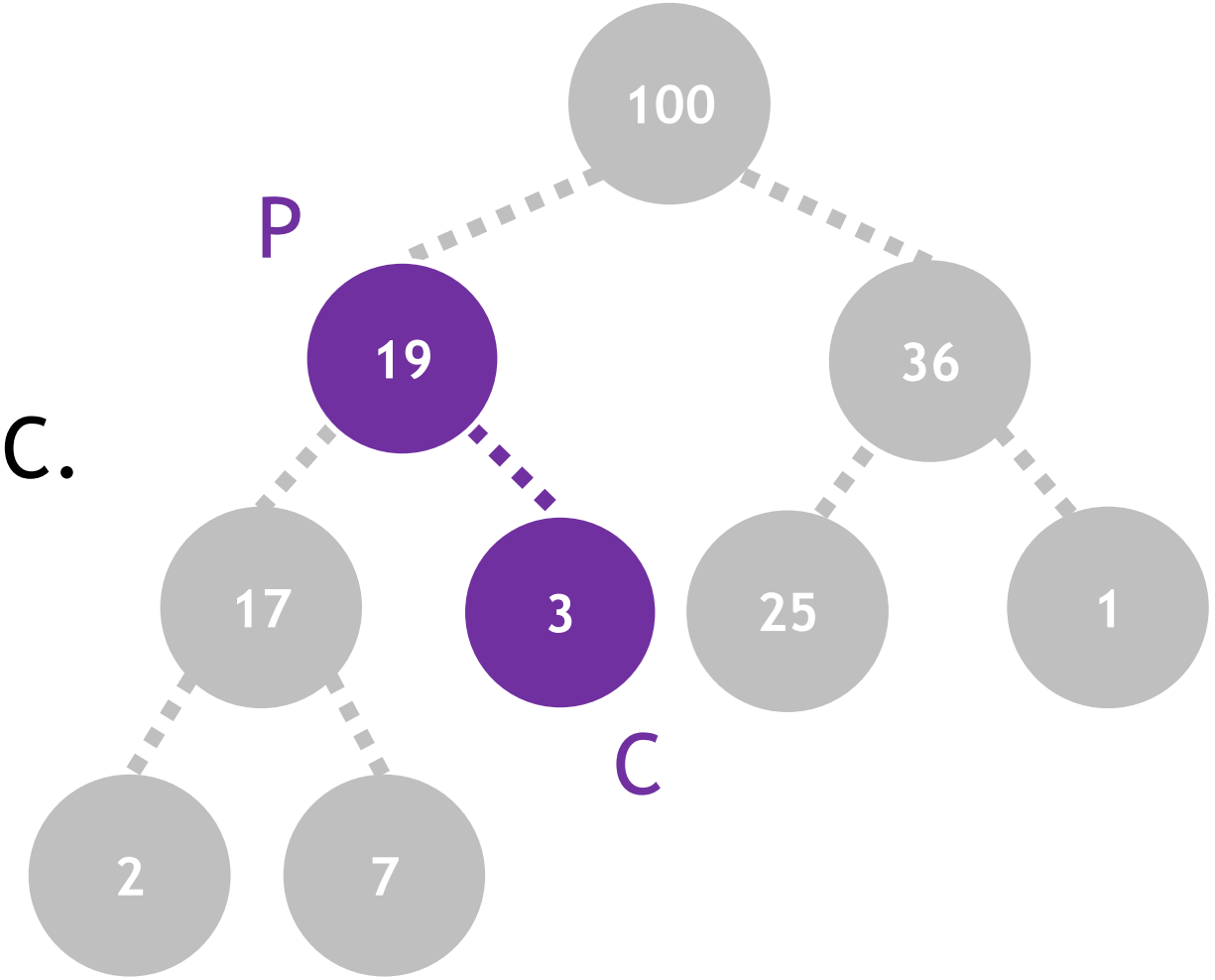
Majme ľubovoľný vrchol C.



Max-Heap

Vlastnosť Max-heap

Majme ľubovoľný vrchol C.
Vrchol P je rodičom C.

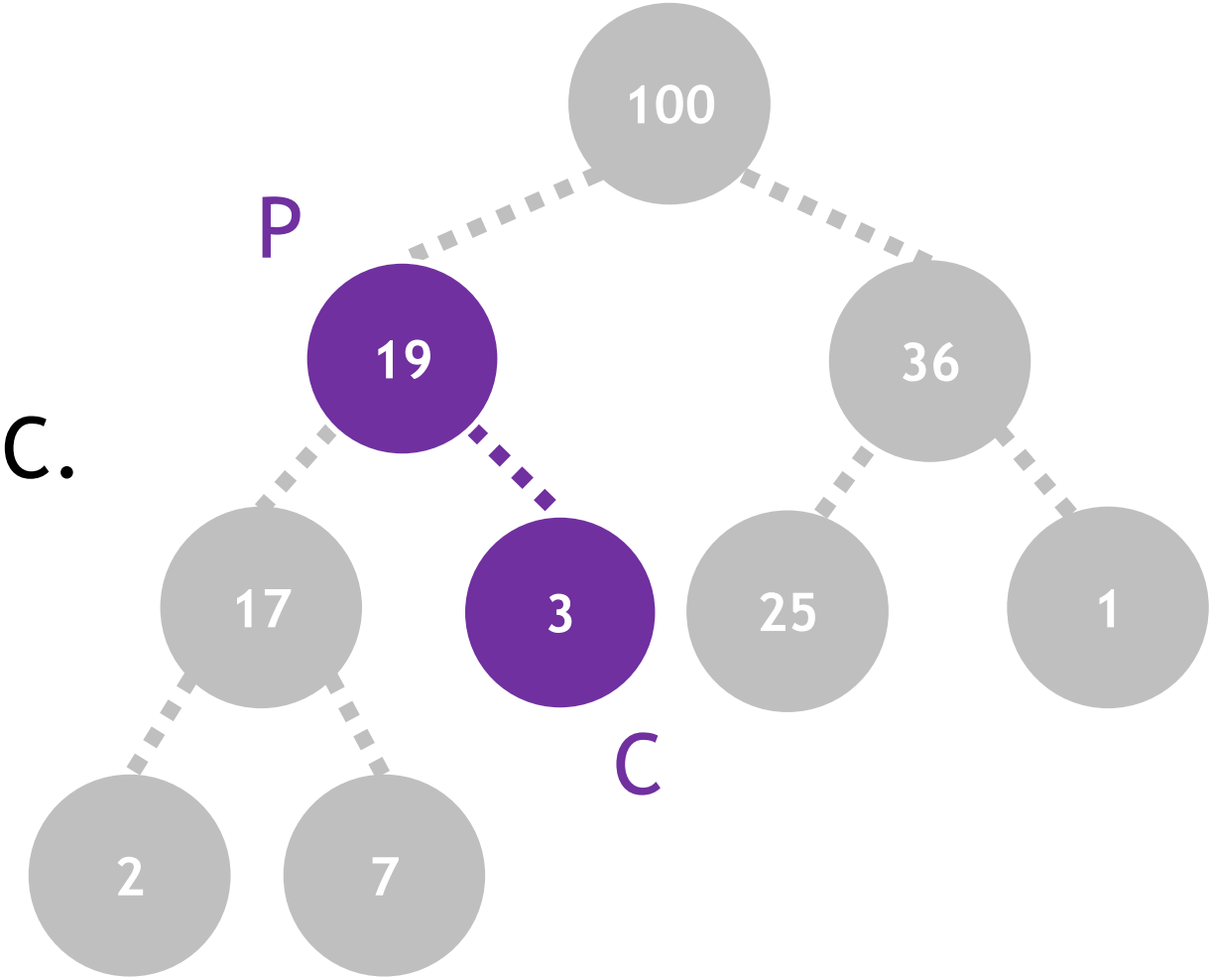


Max-Heap

Vlastnosť Max-heap

Majme ľubovoľný vrchol C.
Vrchol P je rodičom C.

Platí **H(P) ≥ H(C)**

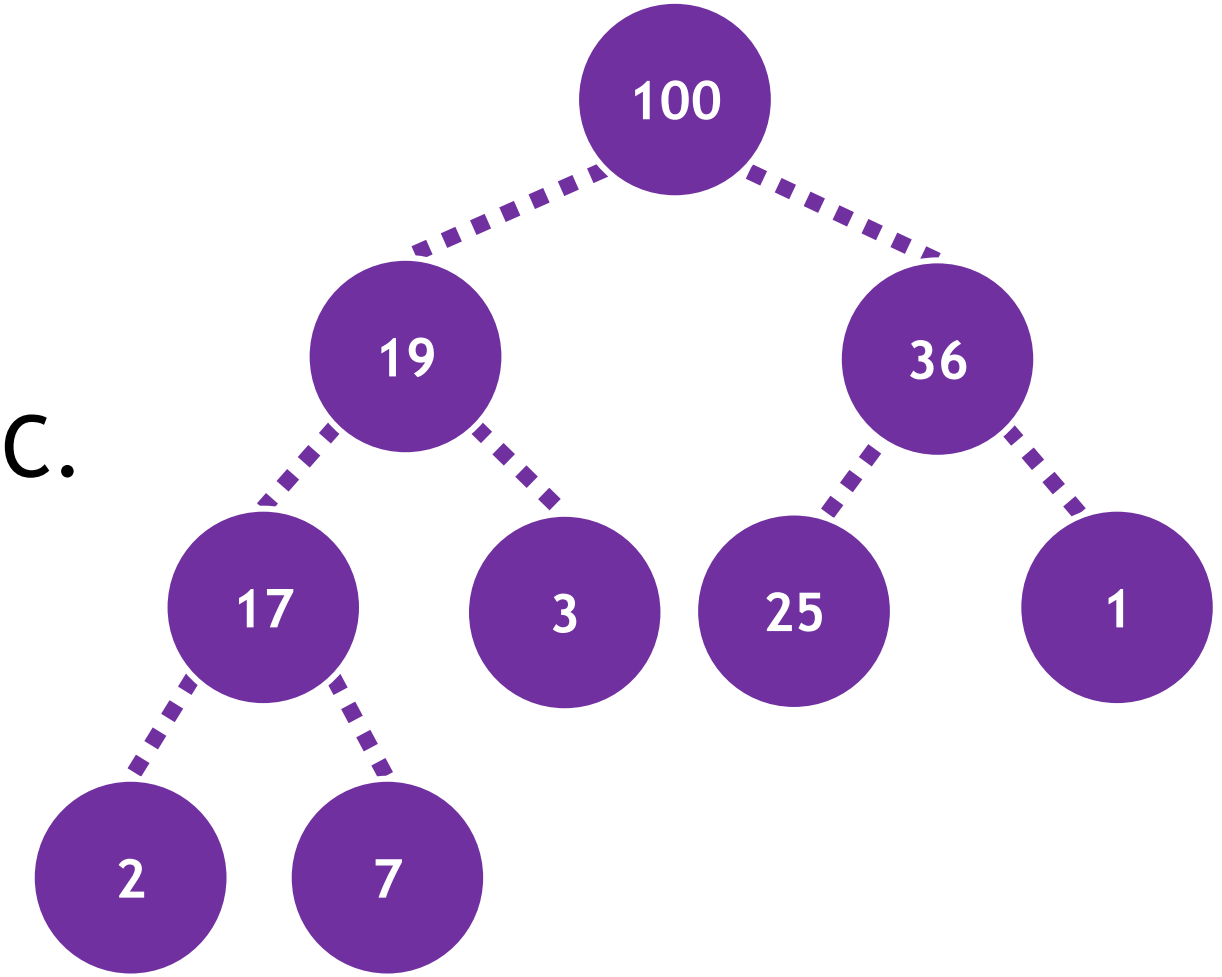


Max-Heap

Vlastnosť Max-heap

Majme ľubovoľný vrchol C.
Vrchol P je rodičom C.

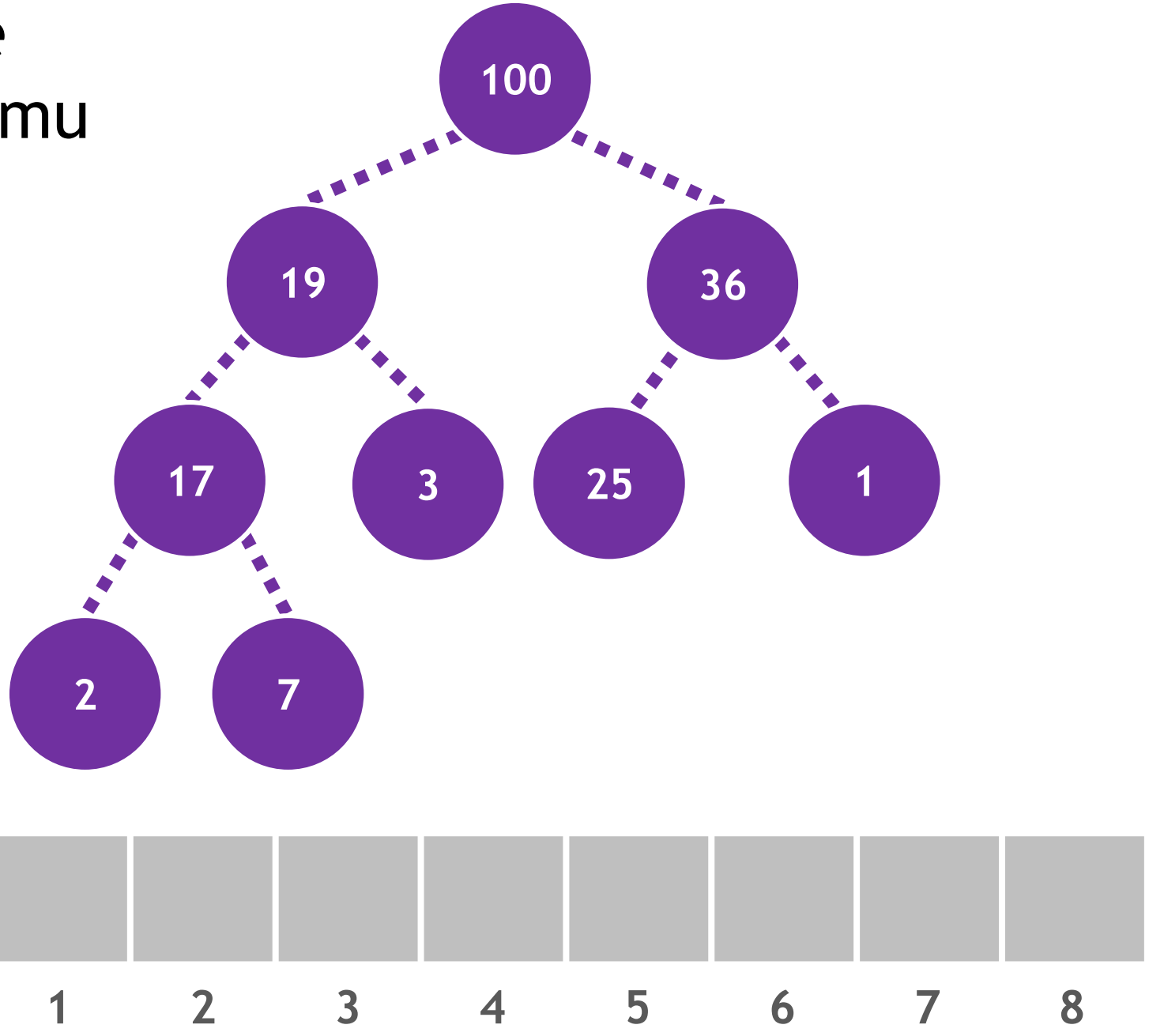
Platí **H(P) ≥ H(C)**

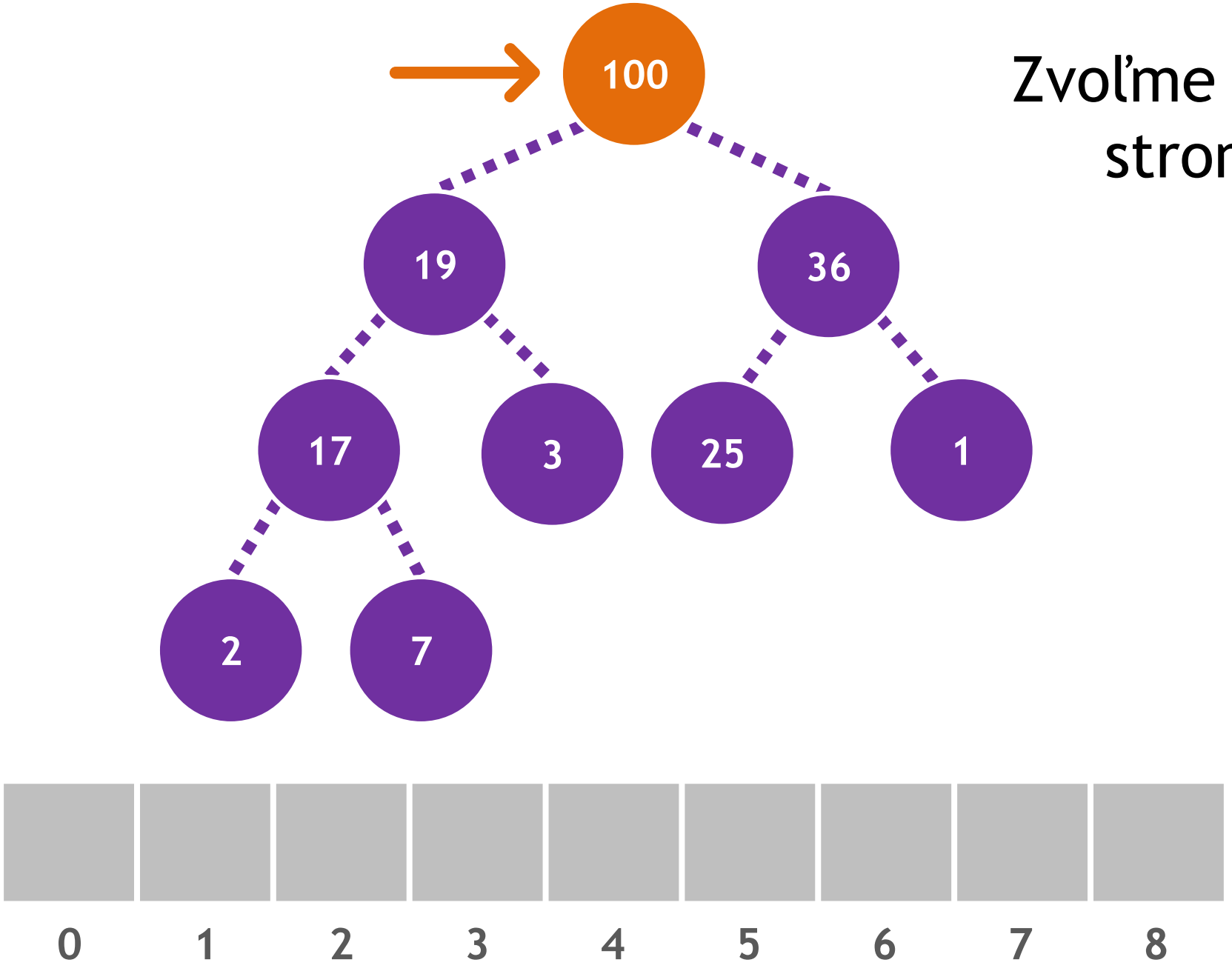


Reprezentácia stromu pomocou poľa

Aby sme mohli triediť pole pomocou algoritmu Heap sort, musíme vedieť **namapovať** vrcholy binárneho (kompletného, doľava zarovnaného) stromu do poľa.

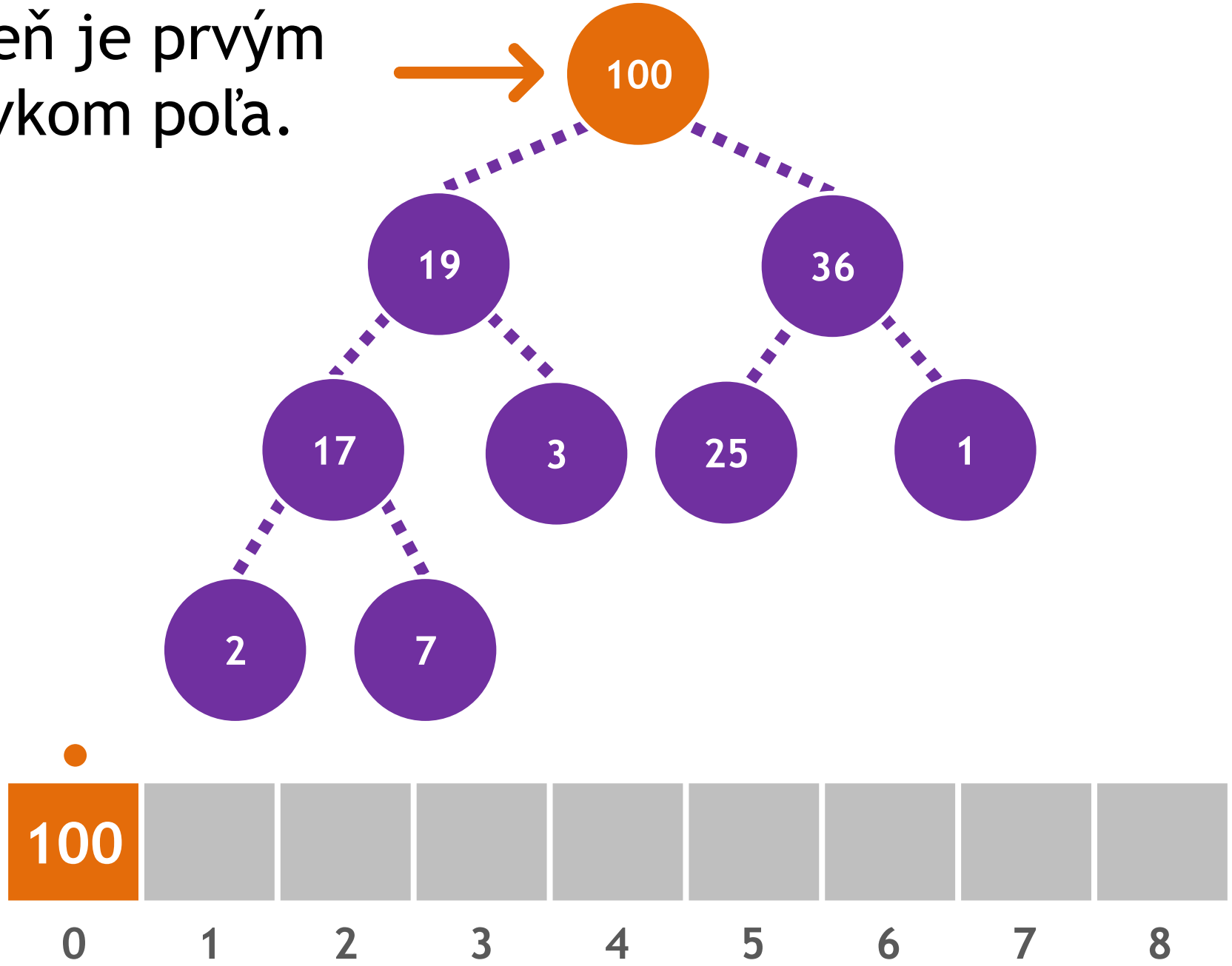
Mapovanie vrcholov stromu do poľa



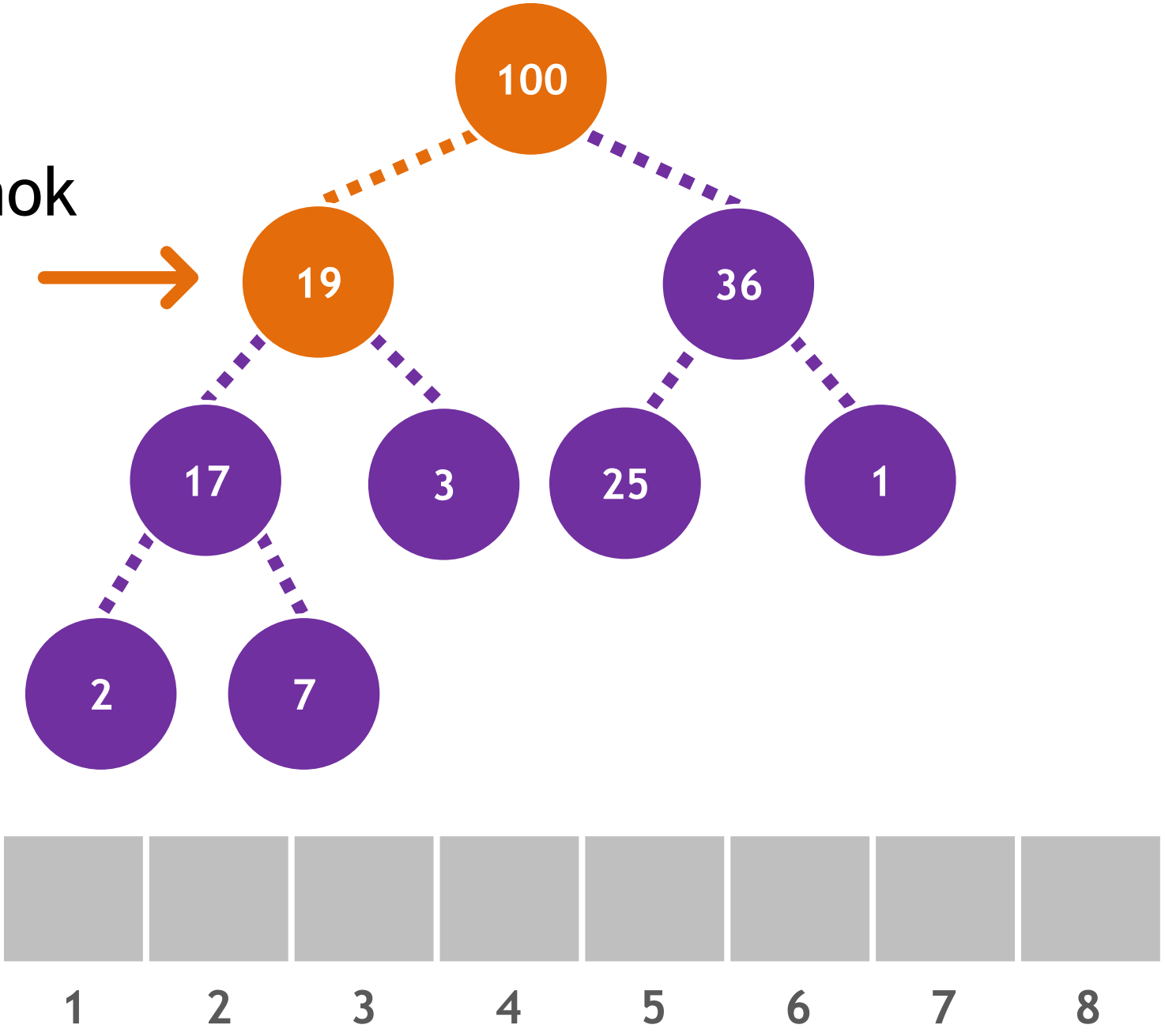


Zvoľme koreň stromu

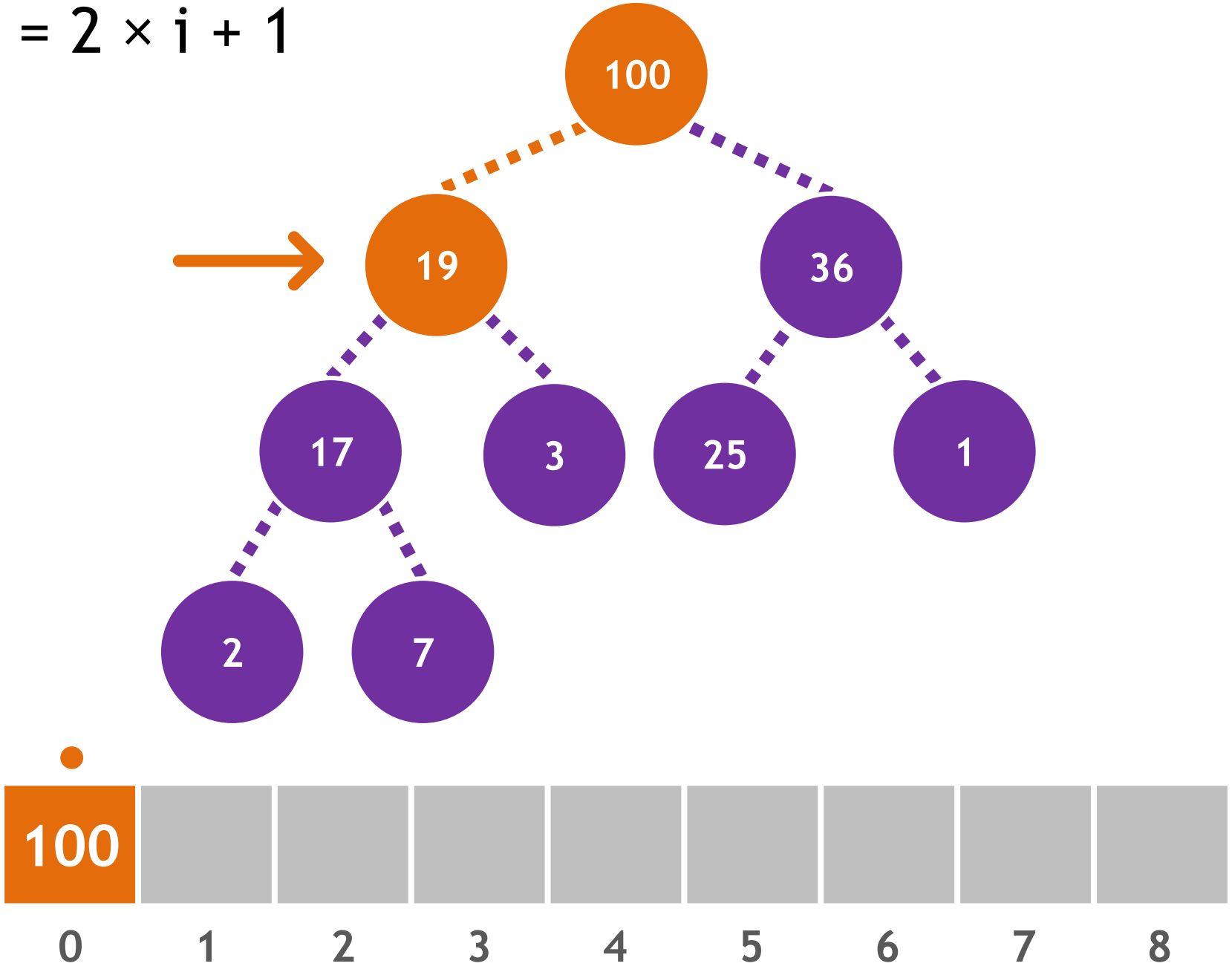
Koreň je prvým prvkom poľa.



Ľavý potomok

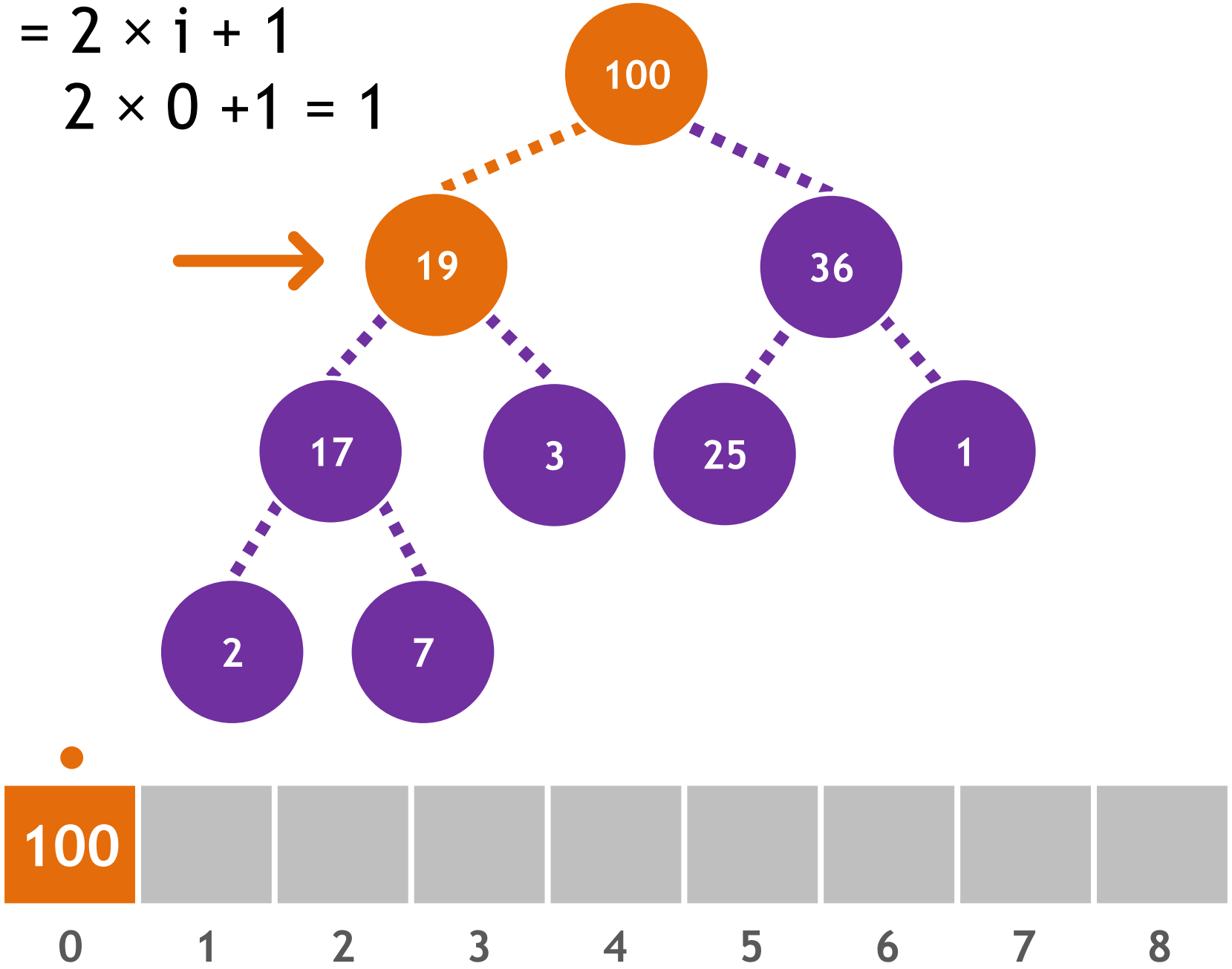


iLeftChild = 2 × i + 1



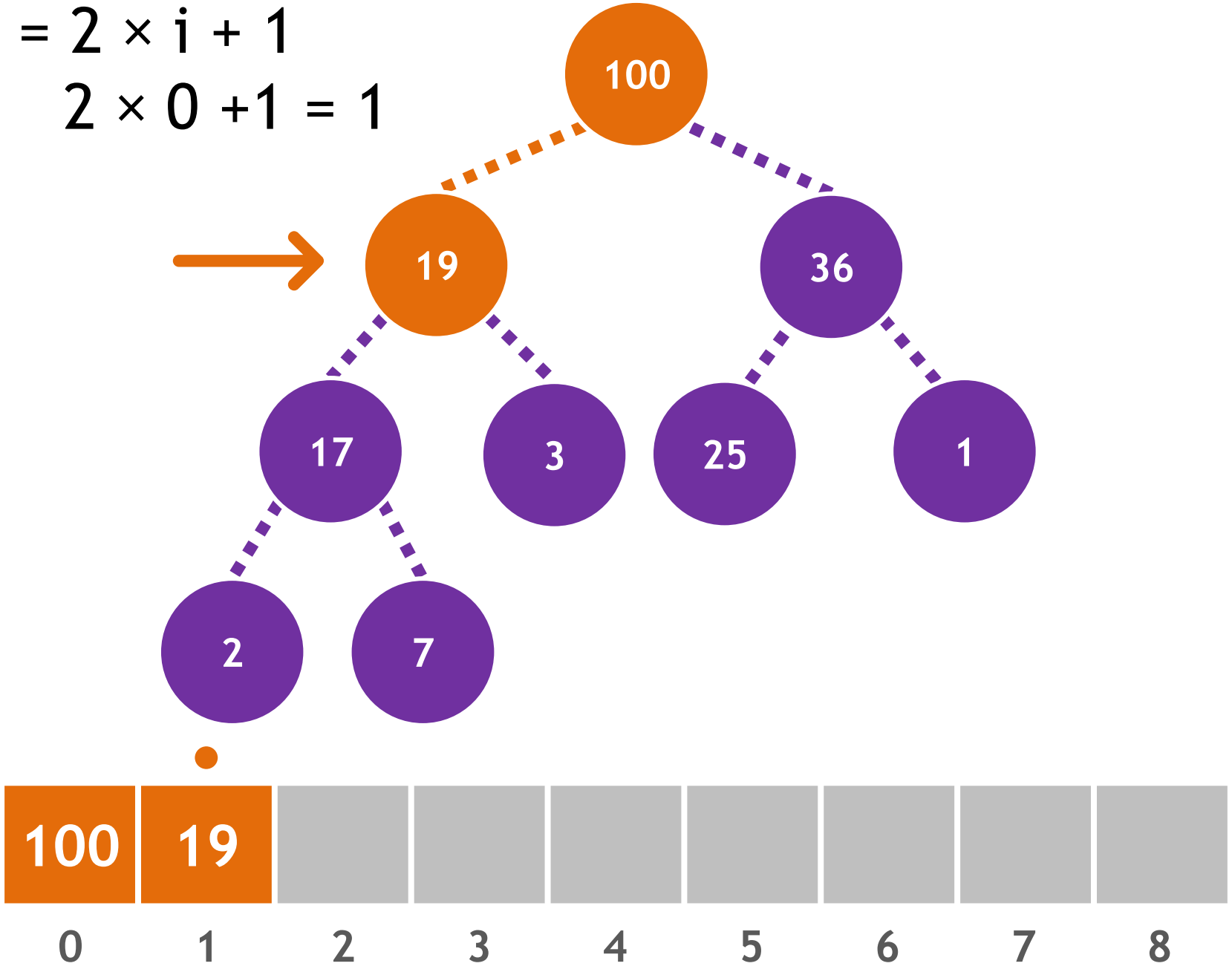
$$iLeftChild = 2 \times i + 1$$

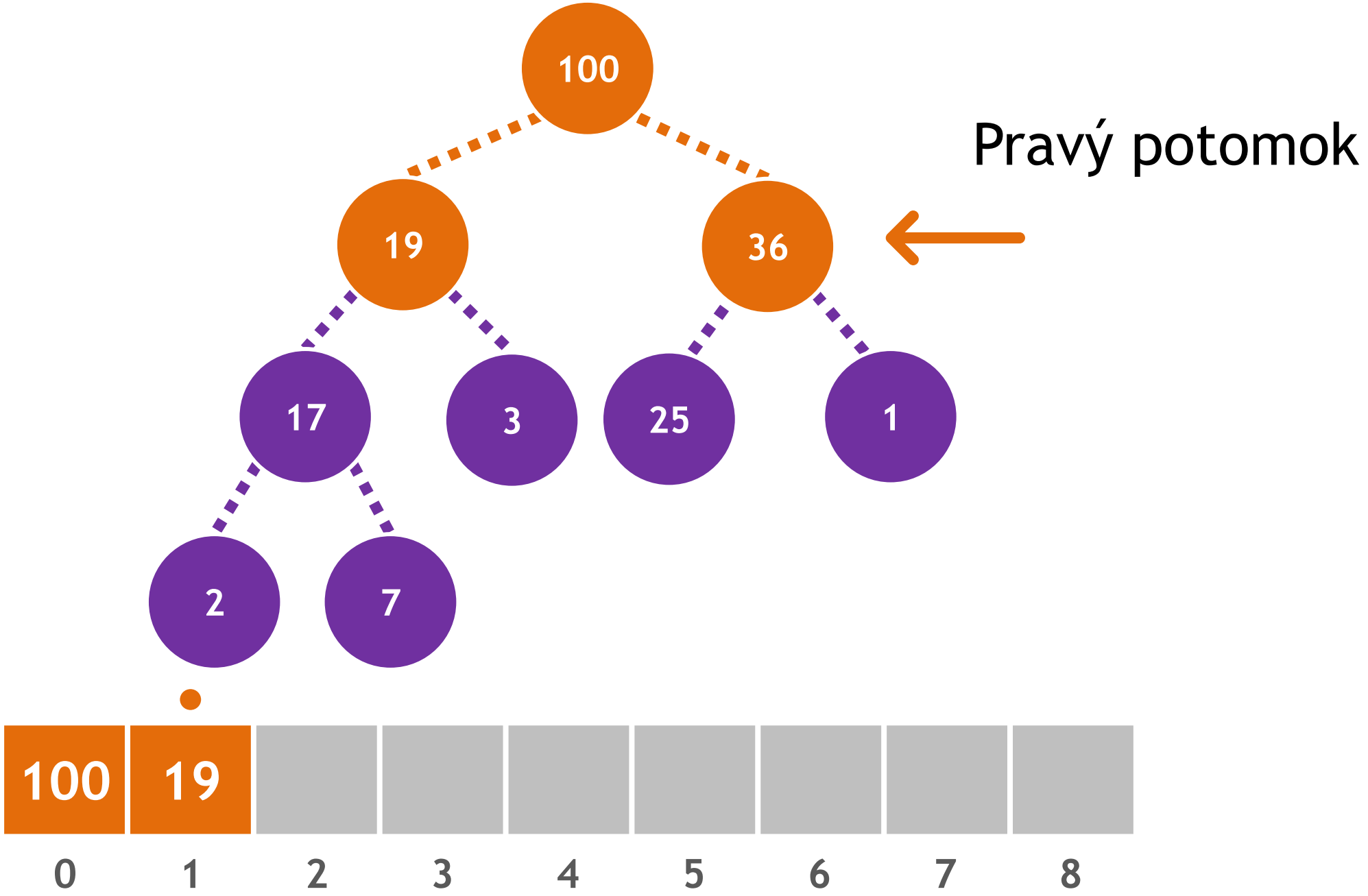
$$2 \times 0 + 1 = 1$$

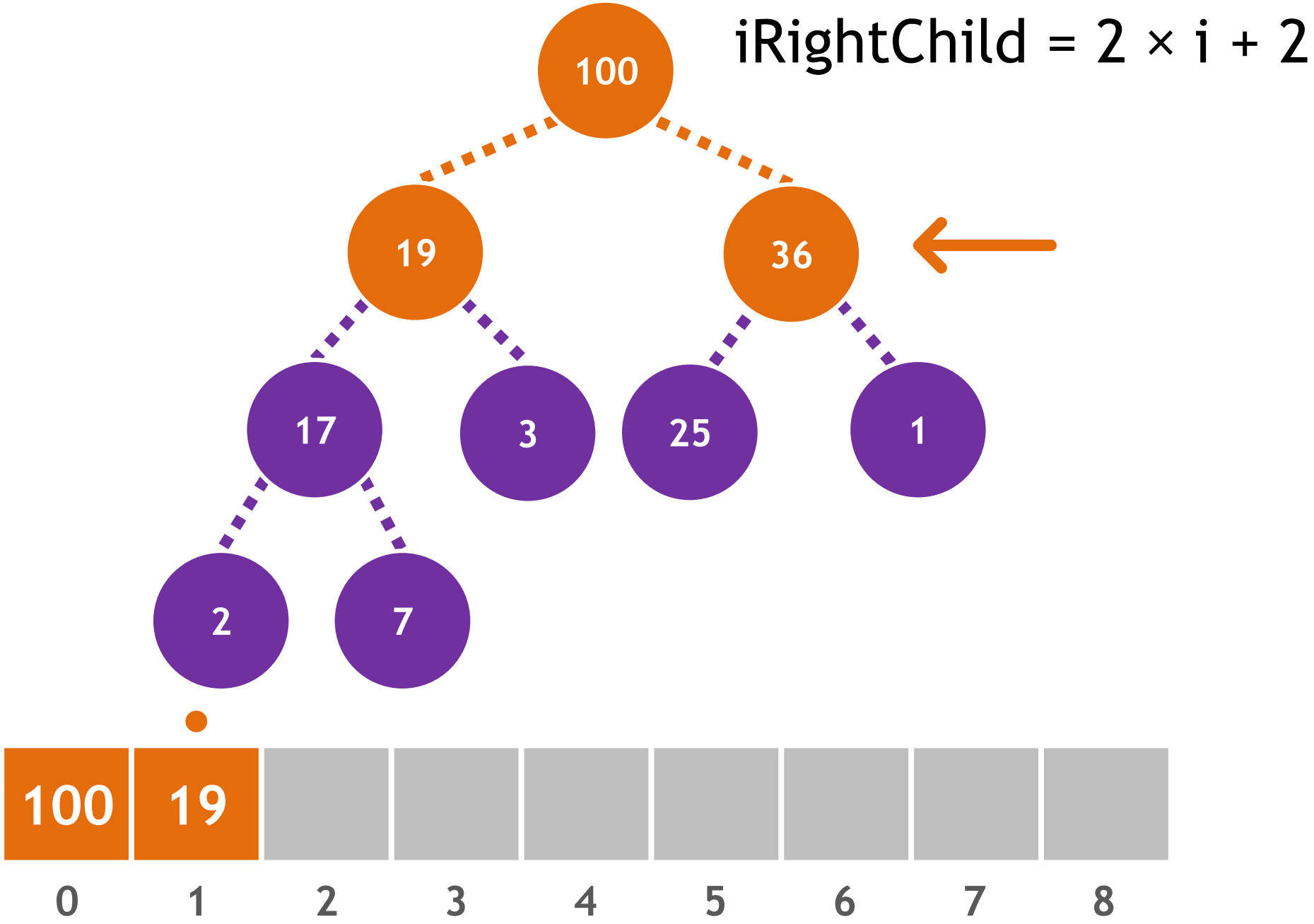


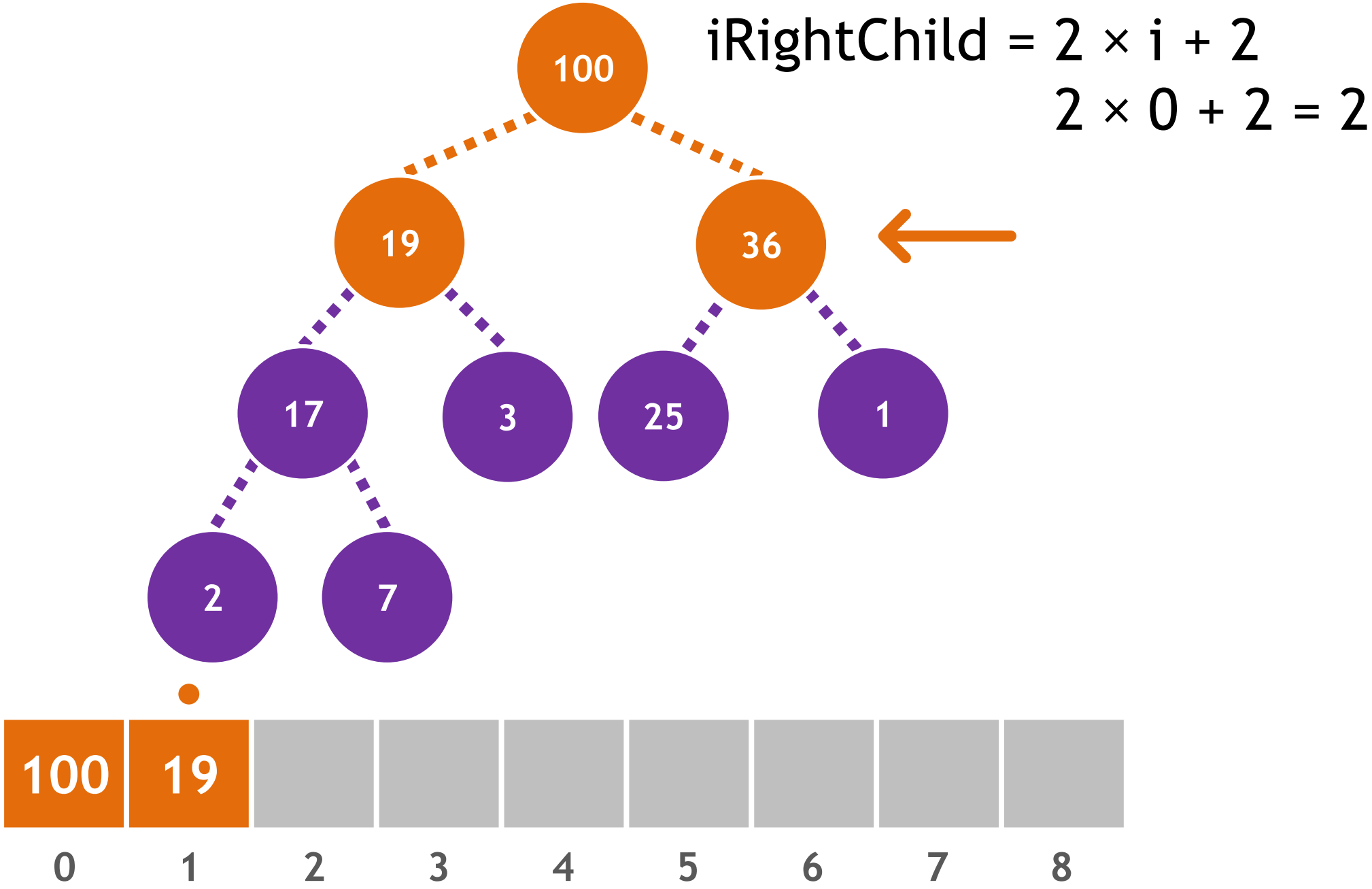
$$iLeftChild = 2 \times i + 1$$

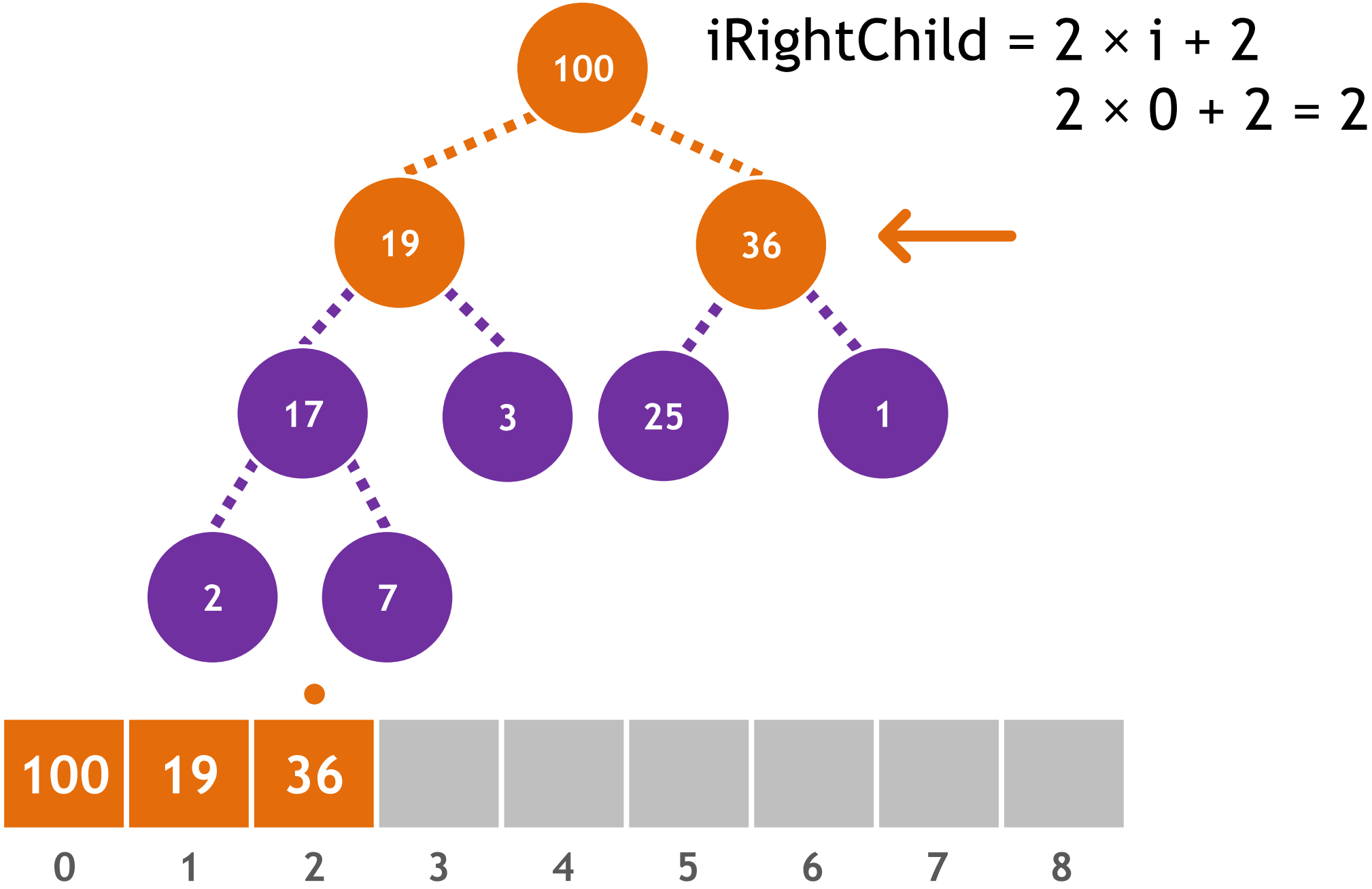
$$2 \times 0 + 1 = 1$$




100
19
36
17
3
25
1
2
7
Pravý potomok
100
19
0
1
2
3
4
5
6
7
8

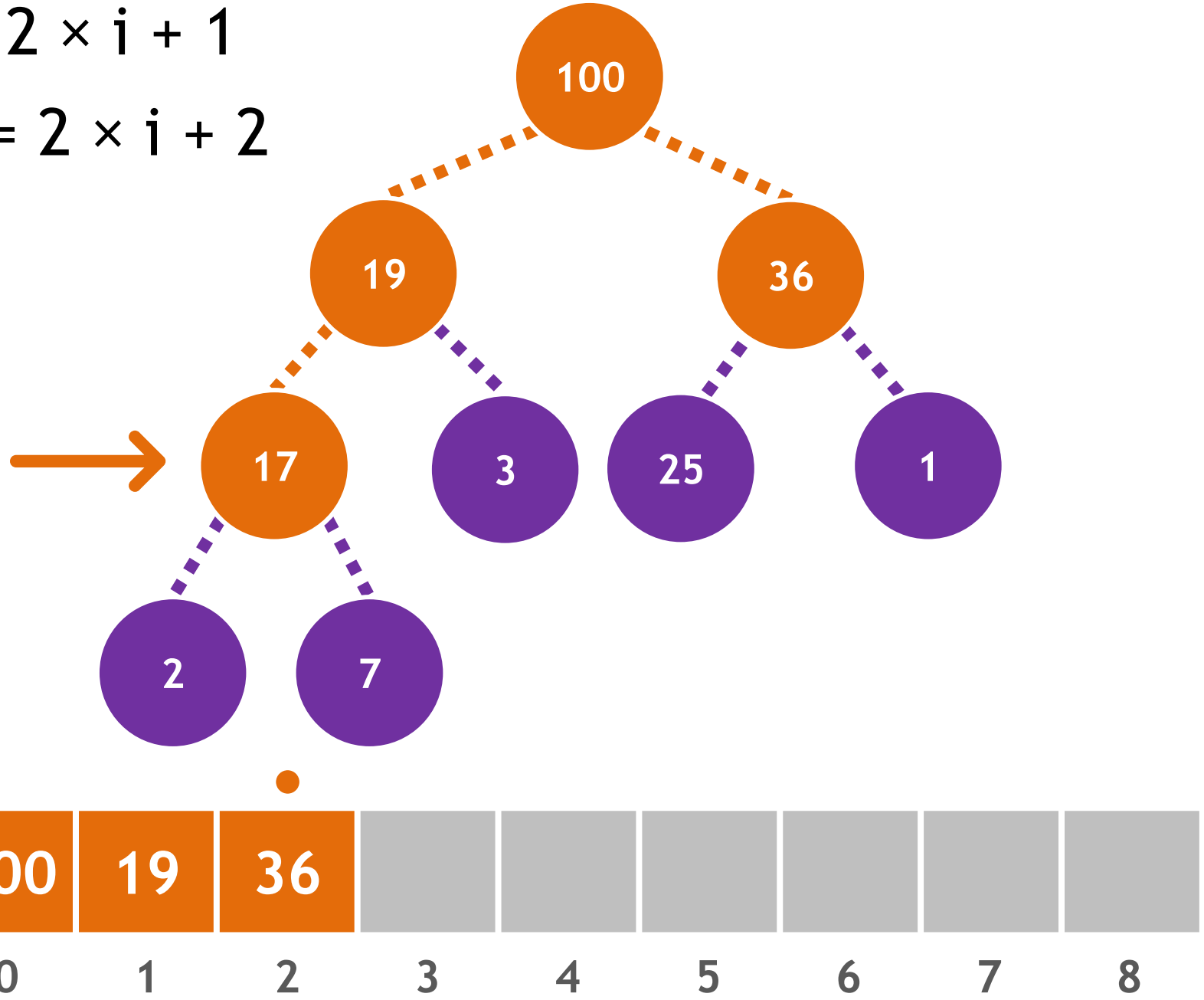

iRightChild = 2 × i + 2
100
19
36
17
3
25
1
2
7
100
19
0
1
2
3
4
5
6
7
8


iRightChild = 2 × i + 2
2 × 0 + 2 = 2
100
19
36
17
3
25
1
2
7
100
19
0
1
2
3
4
5
6
7
8


iRightChild = 2 × i + 2
2 × 0 + 2 = 2
100
19
36
17
3
25
1
2
7
100
19
36
0
1
2
3
4
5
6
7
8

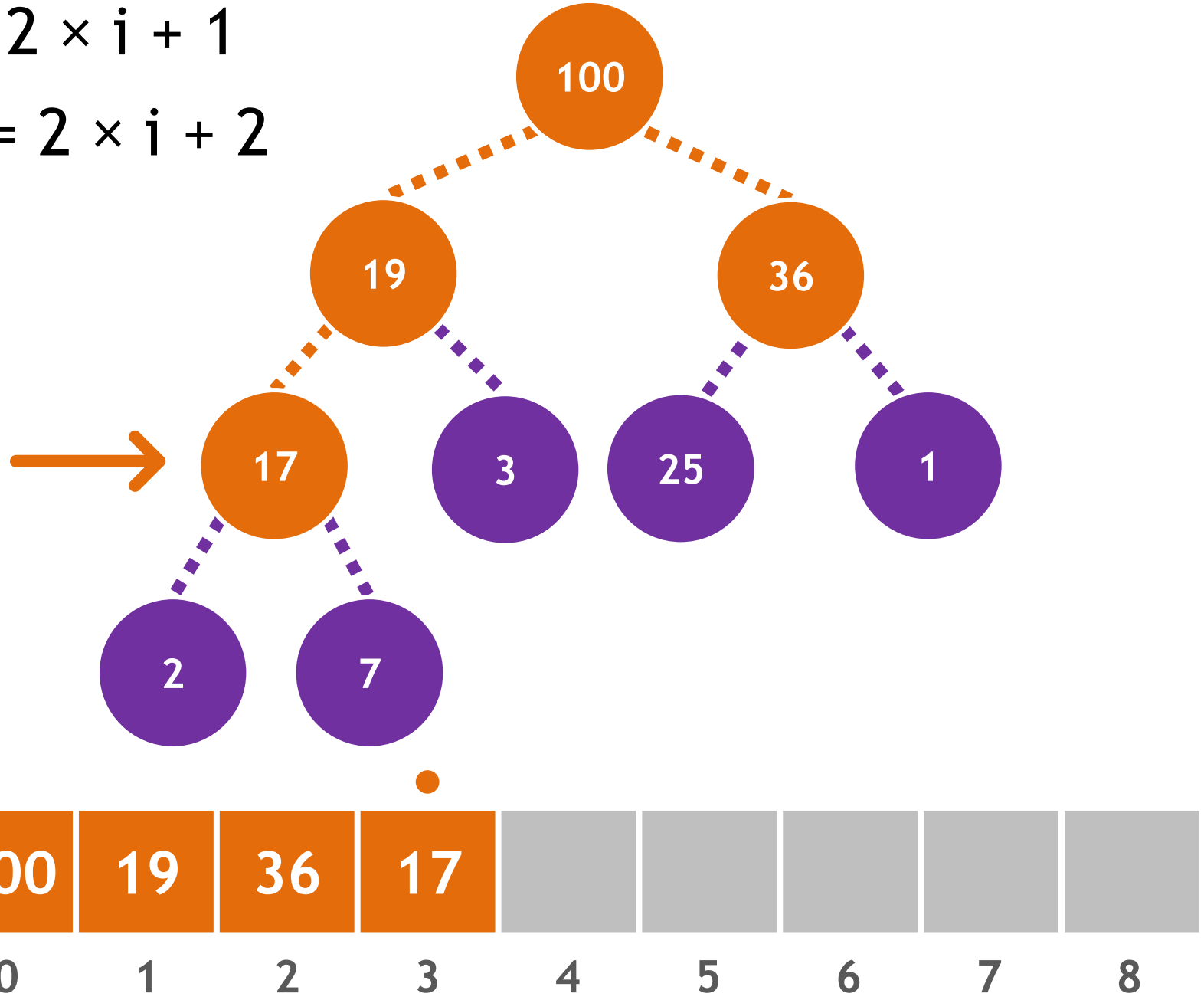
iLeftChild = 2 × i + 1

iRightChild = 2 × i + 2



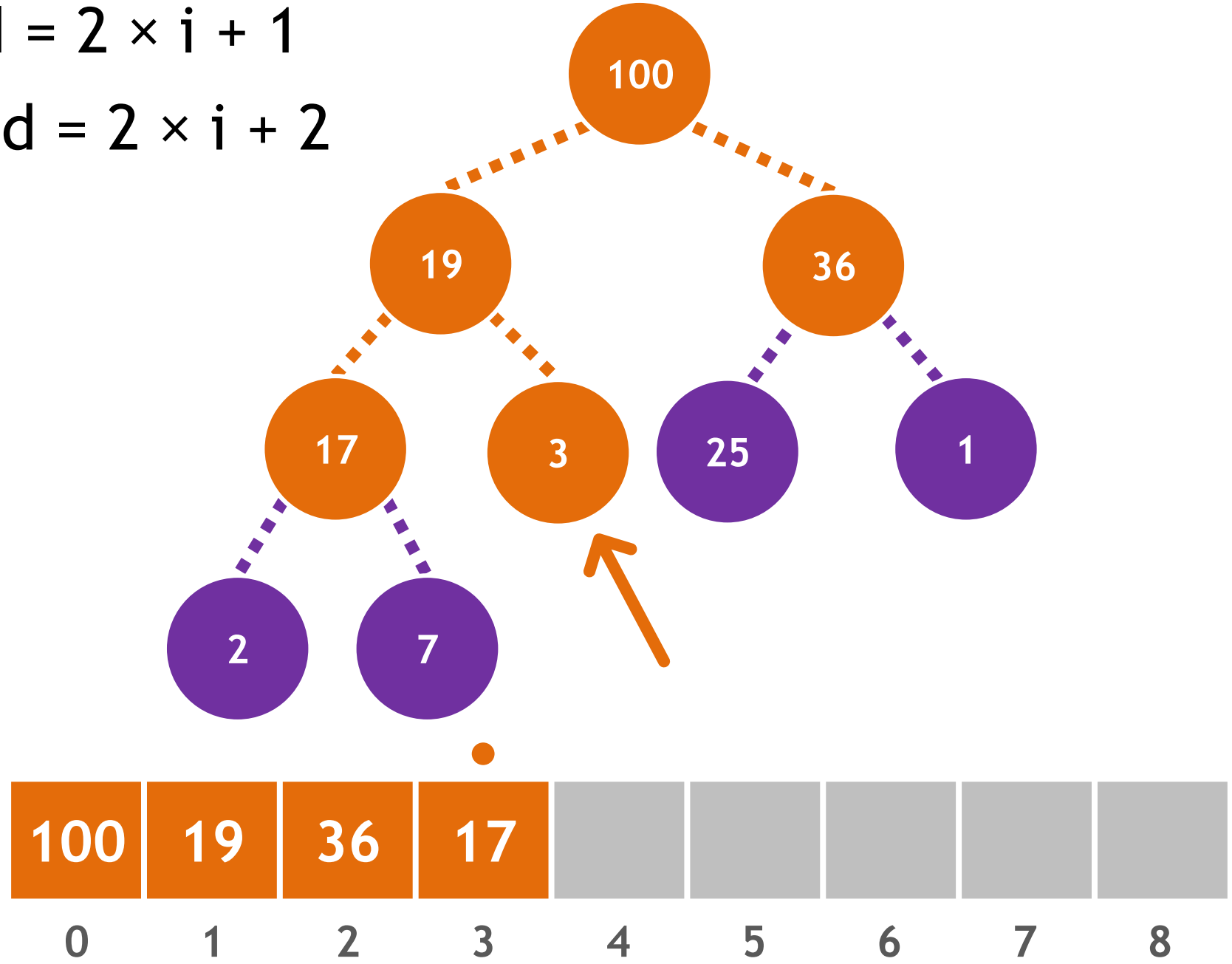
iLeftChild = 2 × i + 1

iRightChild = 2 × i + 2



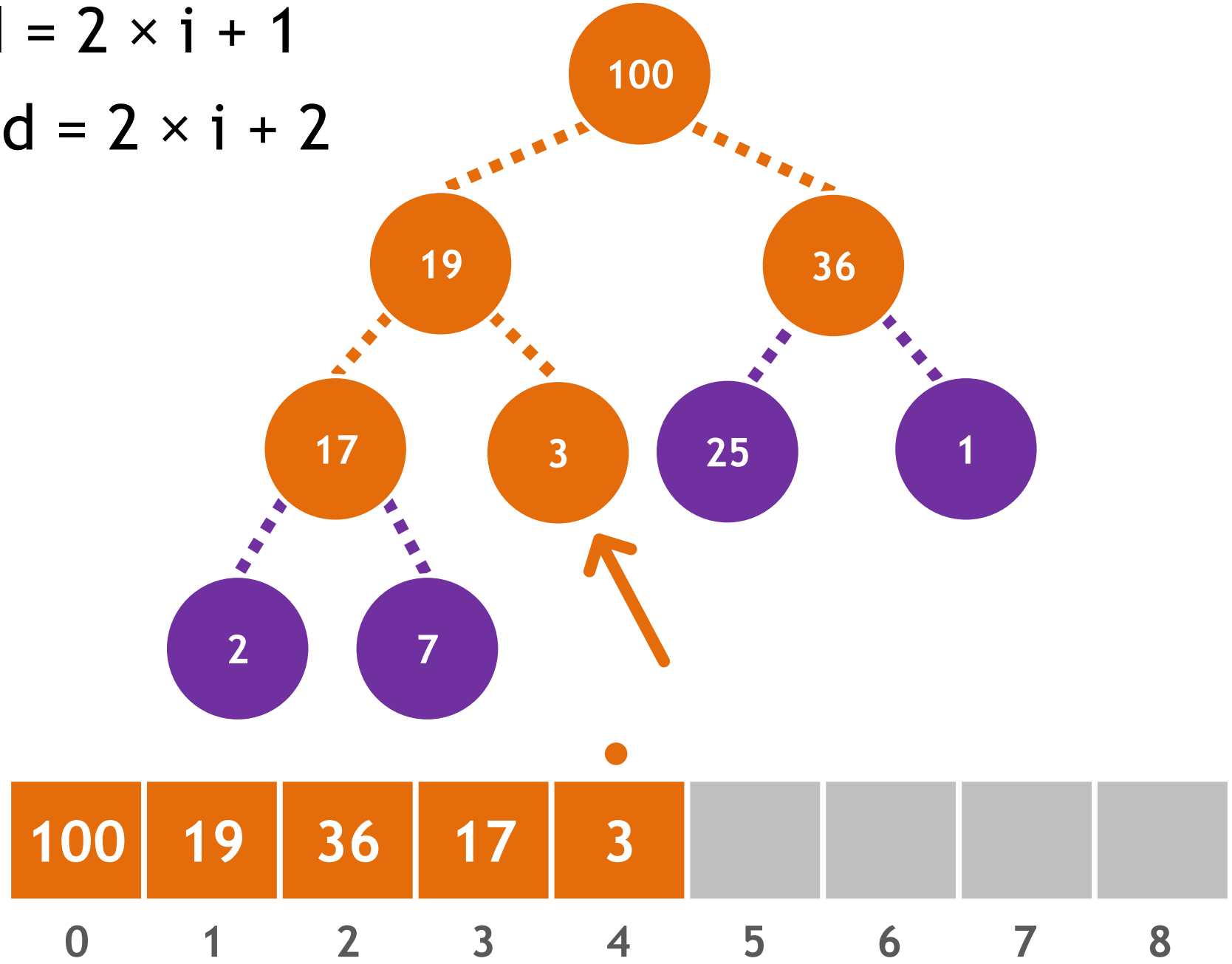
iLeftChild = 2 × i + 1

iRightChild = 2 × i + 2

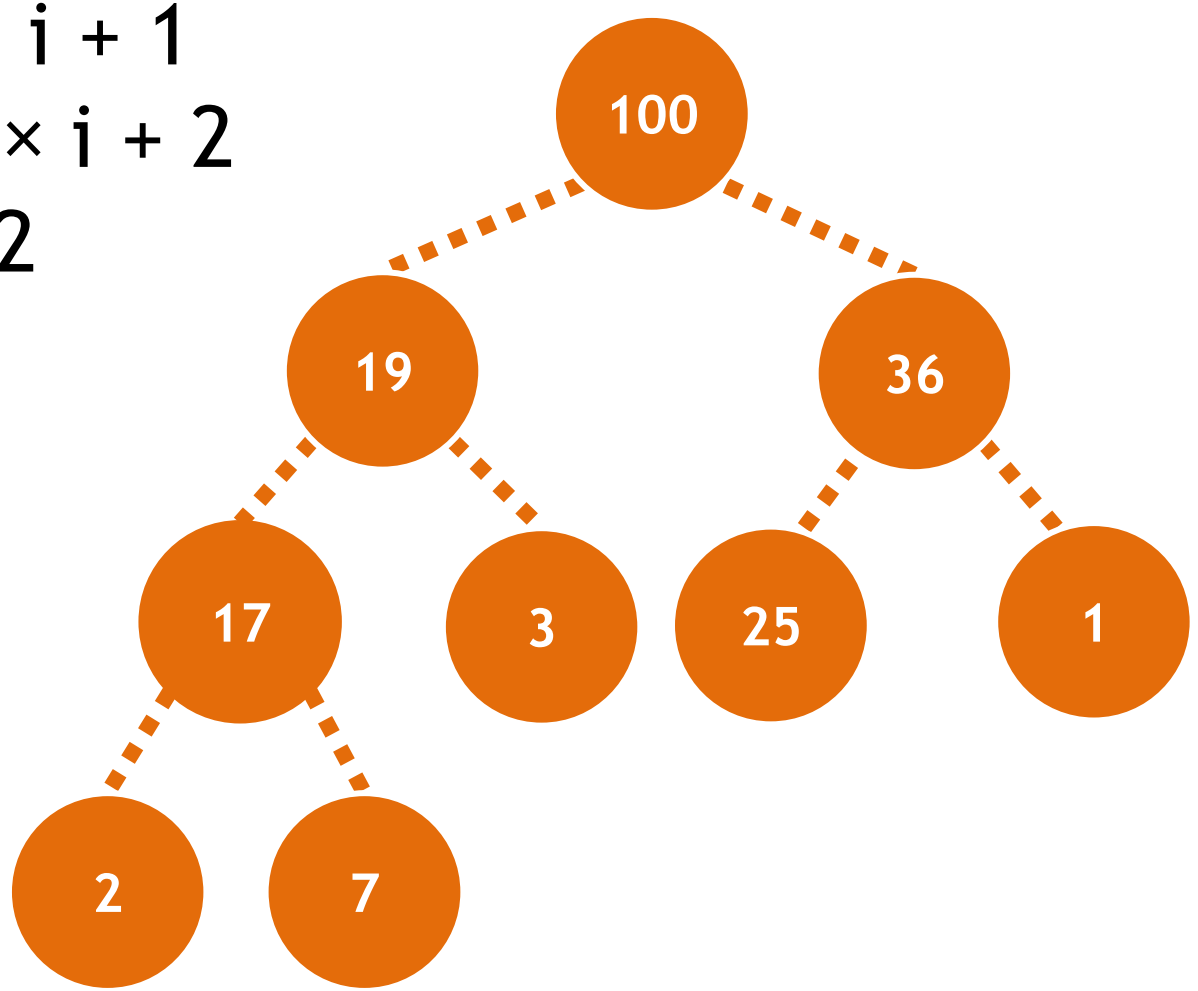


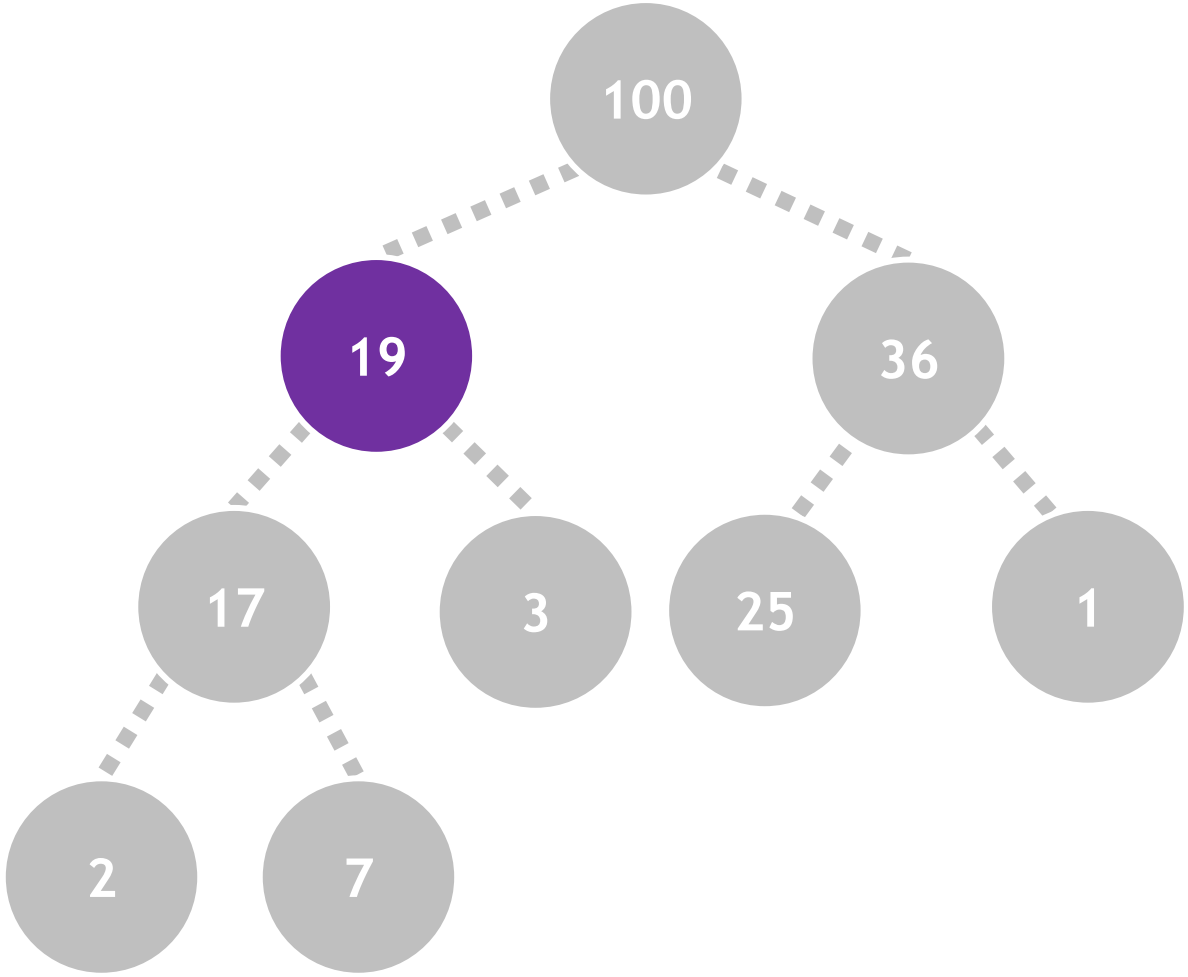
iLeftChild = 2 × i + 1

iRightChild = 2 × i + 2

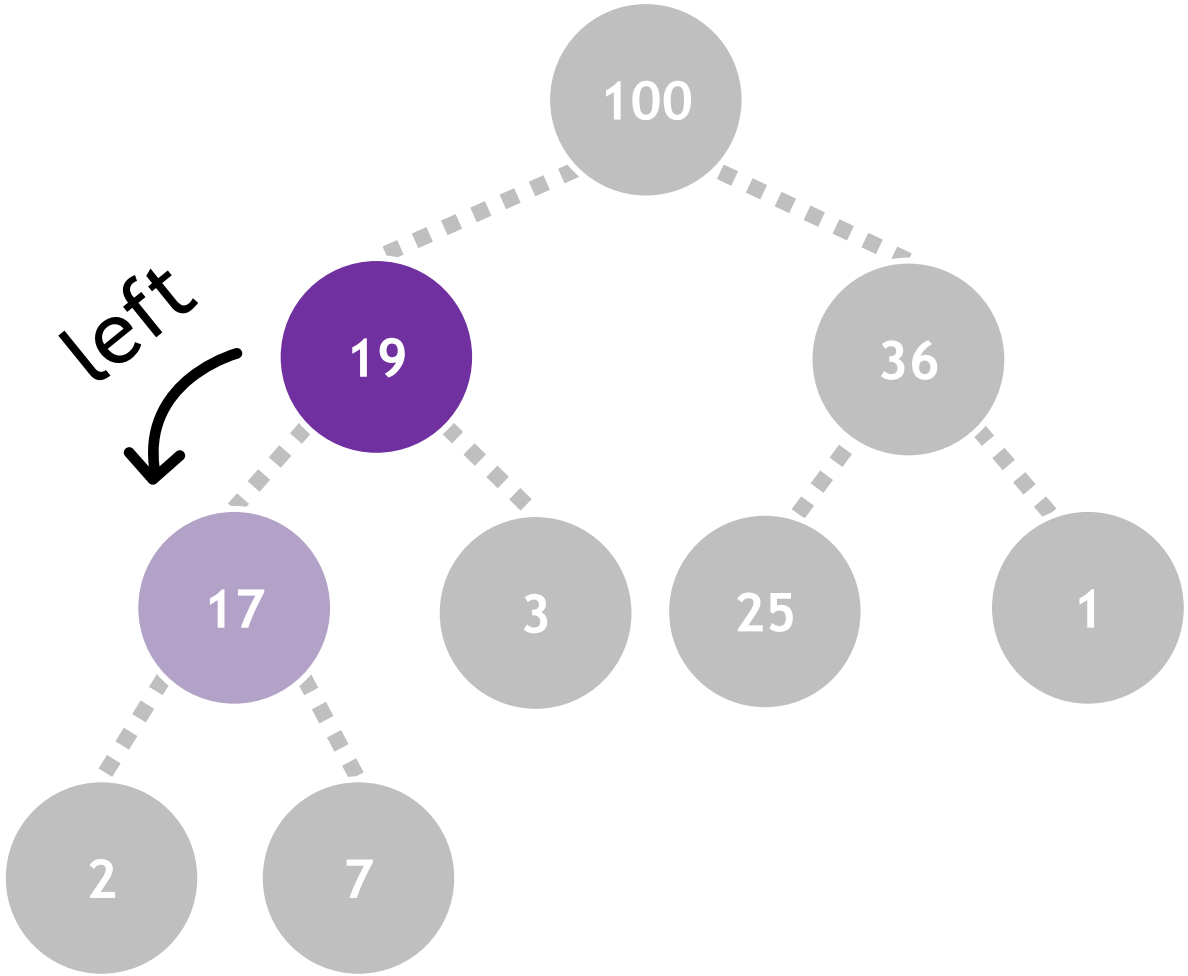


iLeftChild = 2 × i + 1
iRightChild = 2 × i + 2
iParent = (i-1)/2

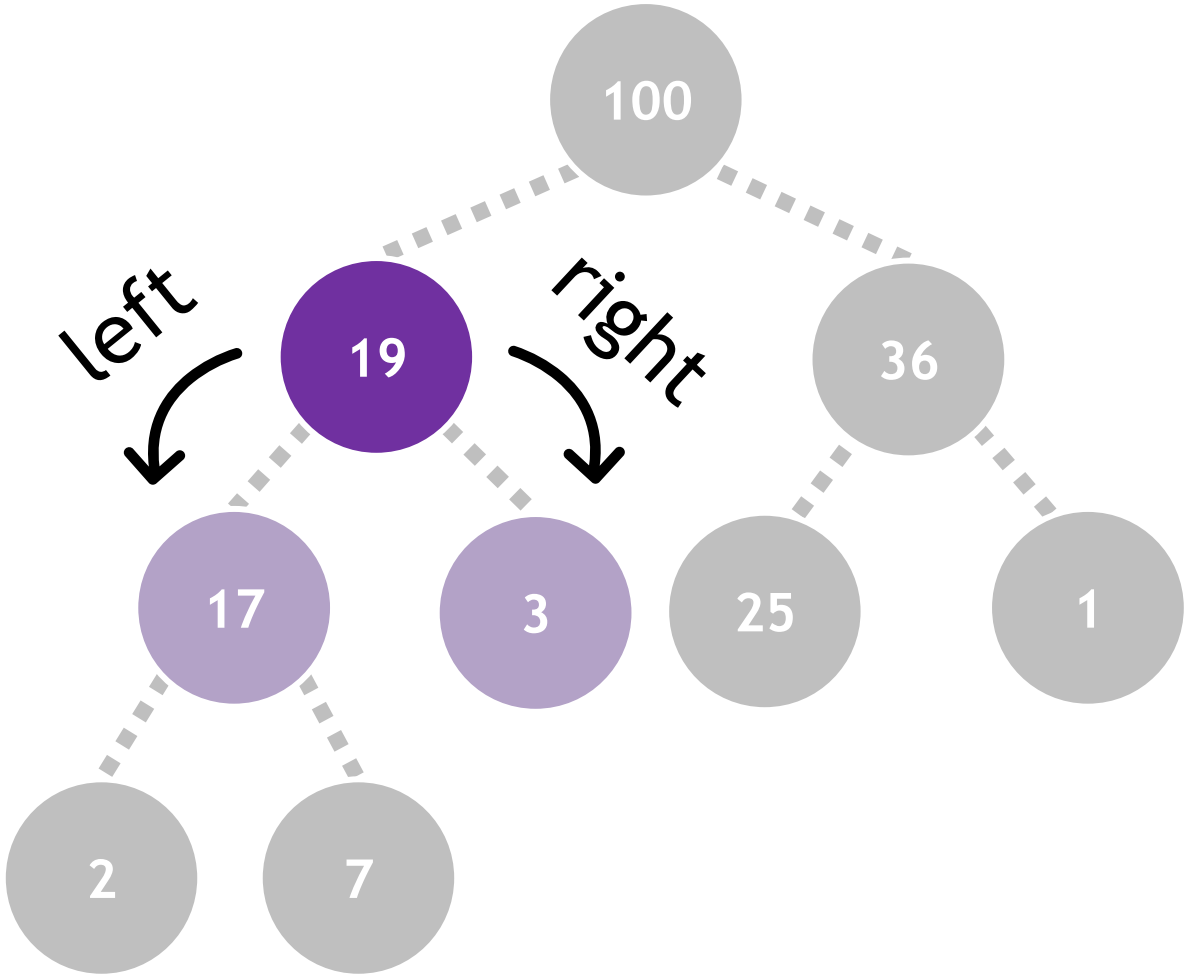



100
19
36
17
3
25
1
2
7

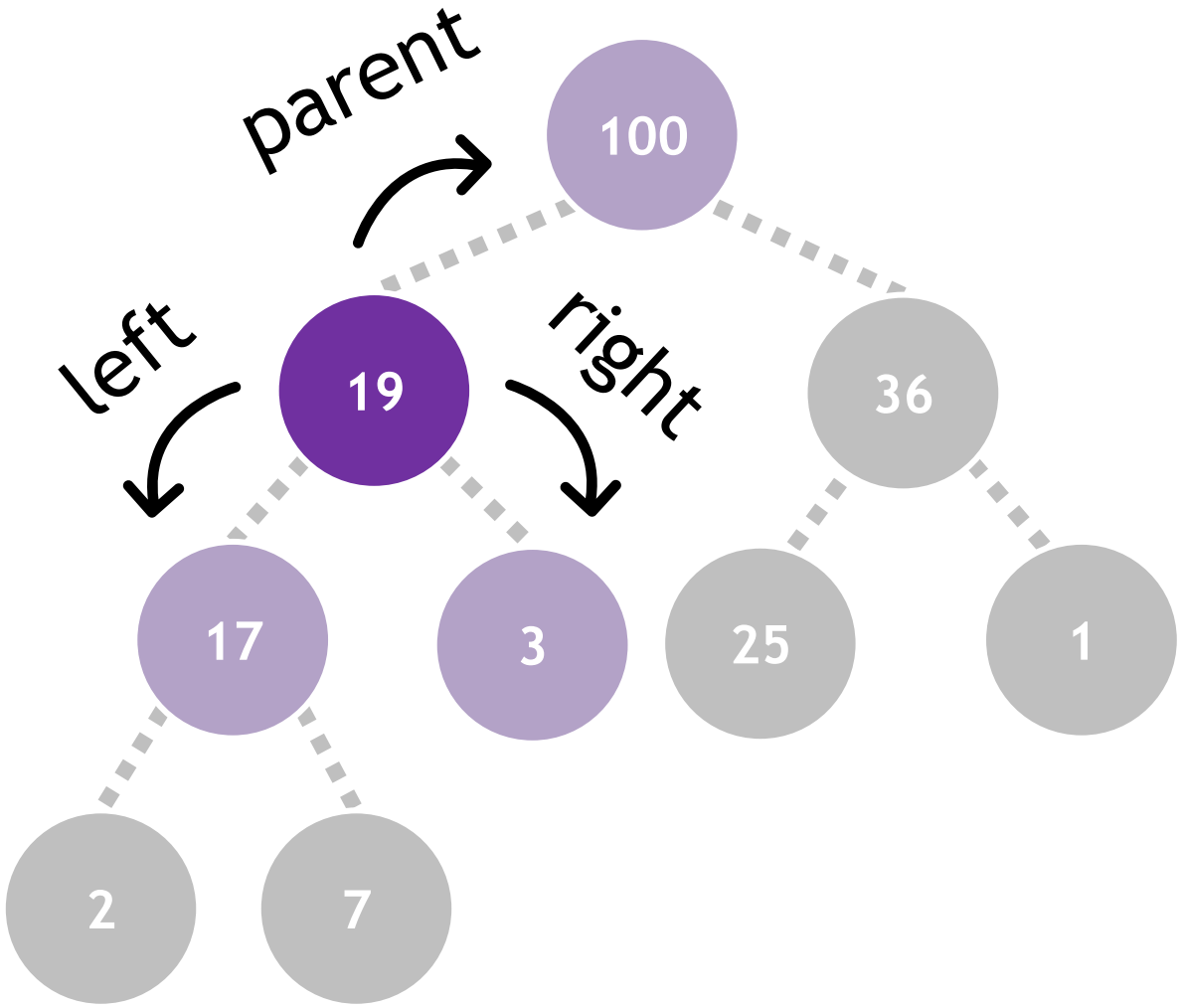
iLeftChild = 2 × i + 1



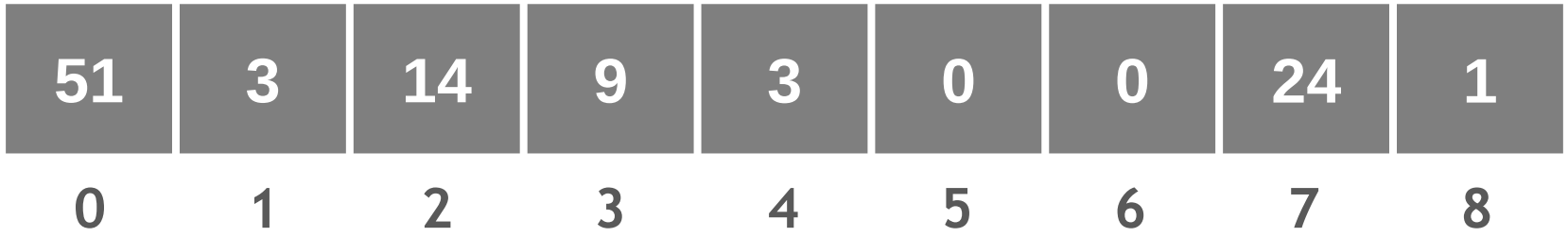
iLeftChild = 2 × i + 1
iRightChild = 2 × i + 2



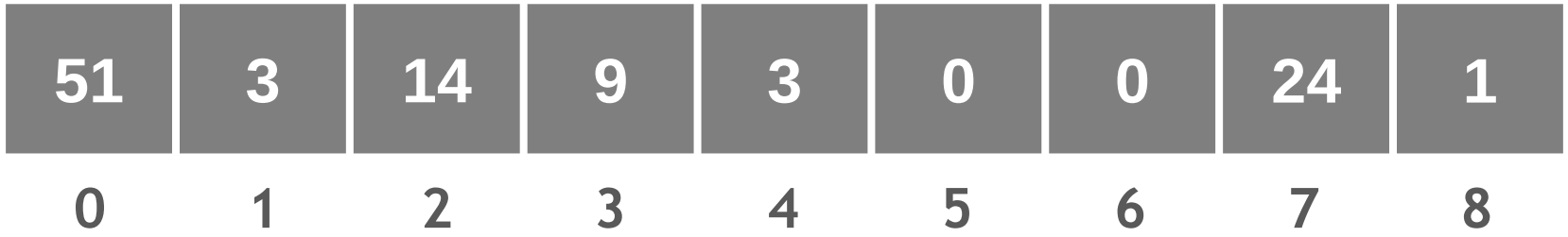
iLeftChild = 2 × i + 1
iRightChild = 2 × i + 2
iParent = (i-1)/2



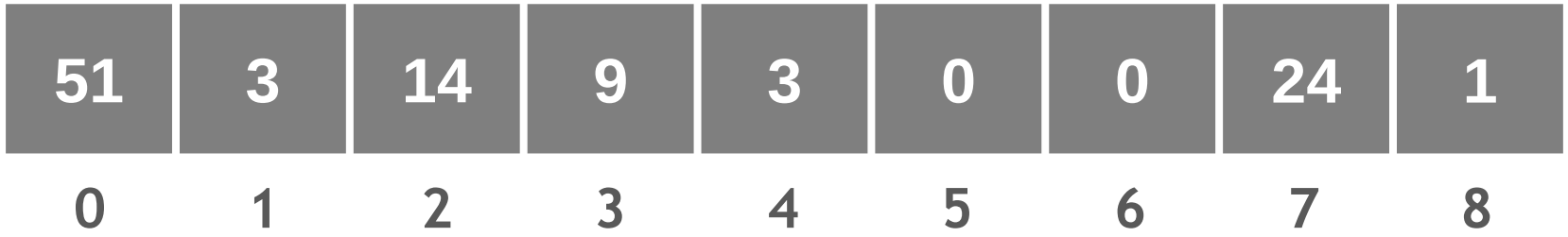
Heap sort vizualizácia


51
3
14
9
3
0
0
24
1
0
1
2
3
4
5
6
7
8

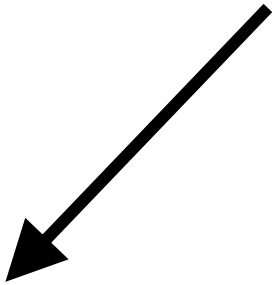
Úloha: zotriediť pole algoritmom Heap sort (vzostupne)

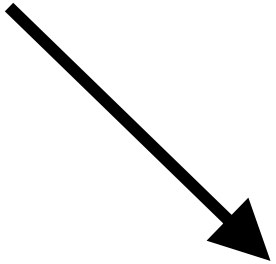


1. Fáza Vytvorenie Max-heapu



1. Fáza Vytvorenie Max-heapu





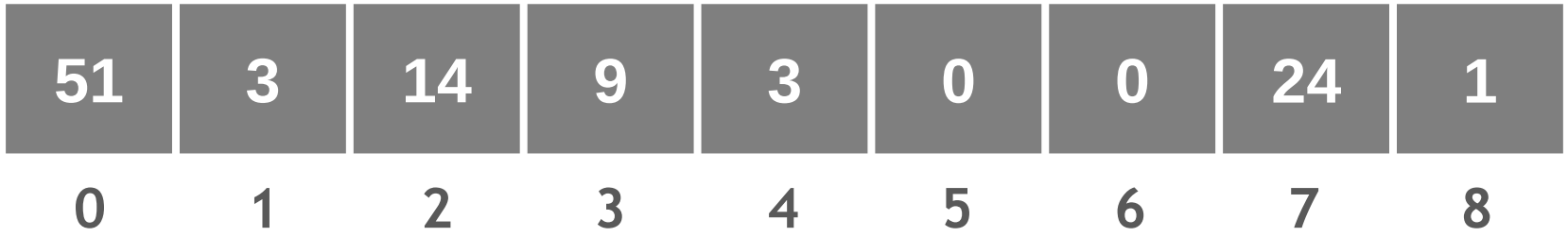
Pomocou operácie **siftUp**

Pomocou operácie **siftDown**

51	3	14	9	3	0	0	24	1
0	1	2	3	4	5	6	7	8

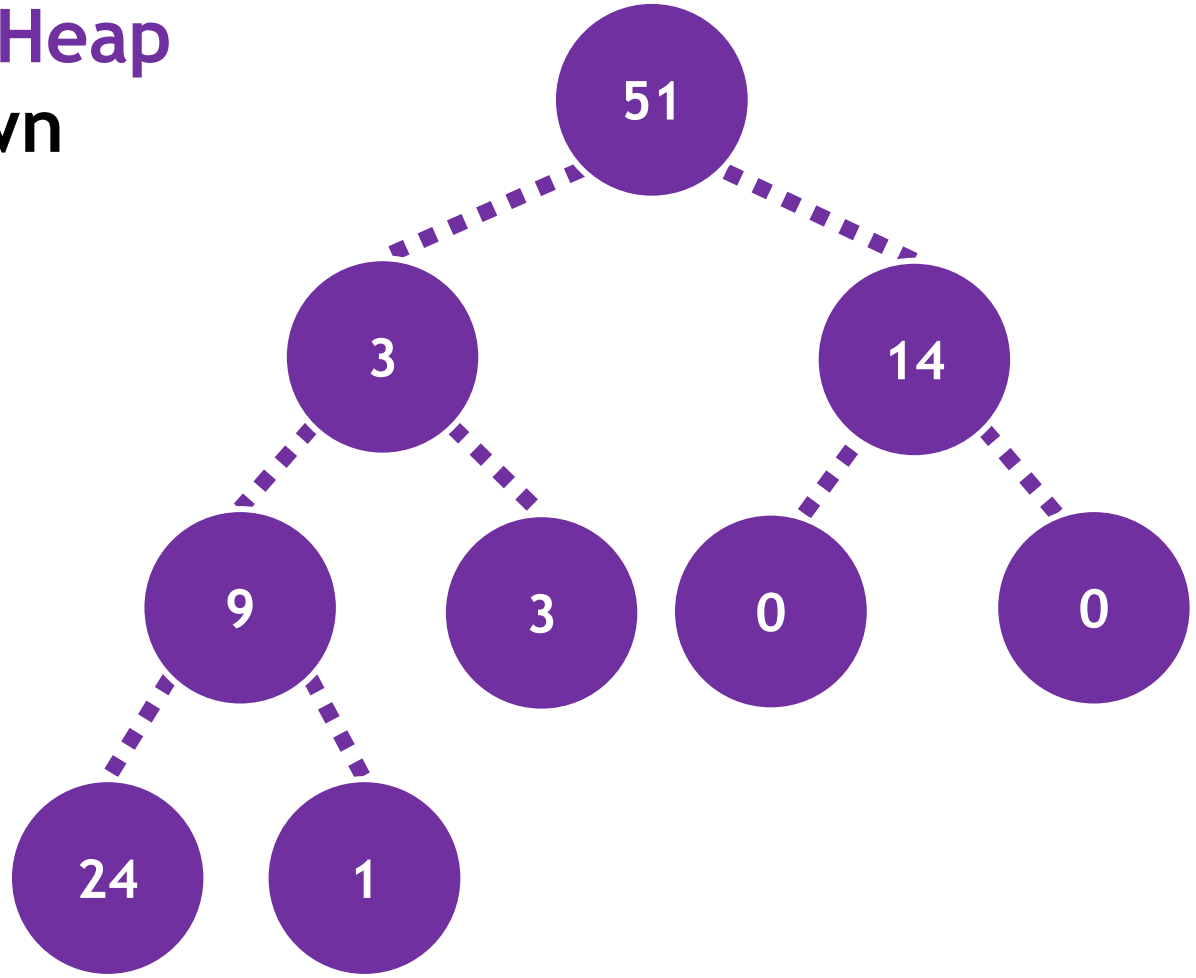
Vytvárame Max-Heap

Pomocou operácie **siftDown**



Vytvárame Max-Heap

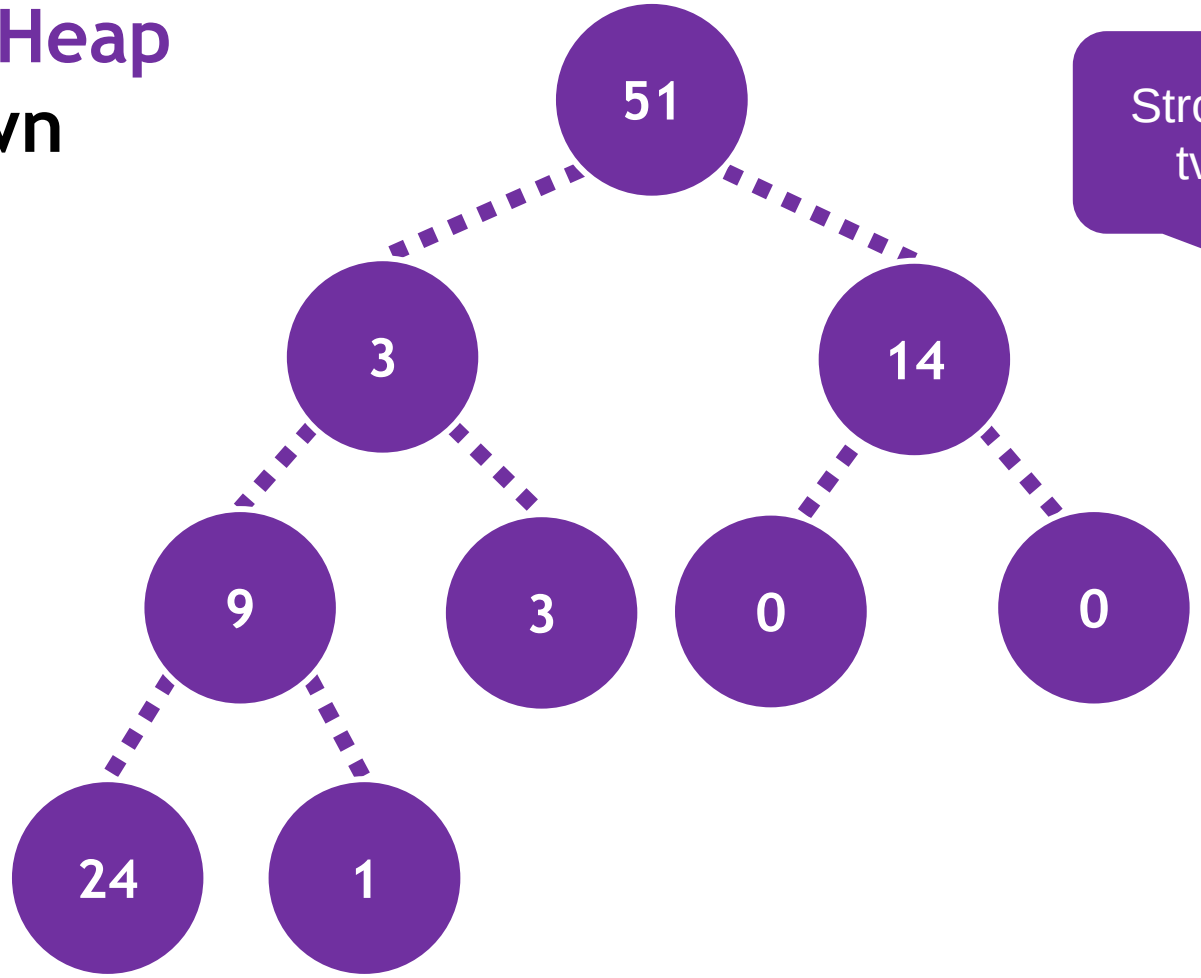
Operácia **siftDown**



51	3	14	9	3	0	0	24	1
0	1	2	3	4	5	6	7	8

Vytvárame Max-Heap

Operácia **siftDown**

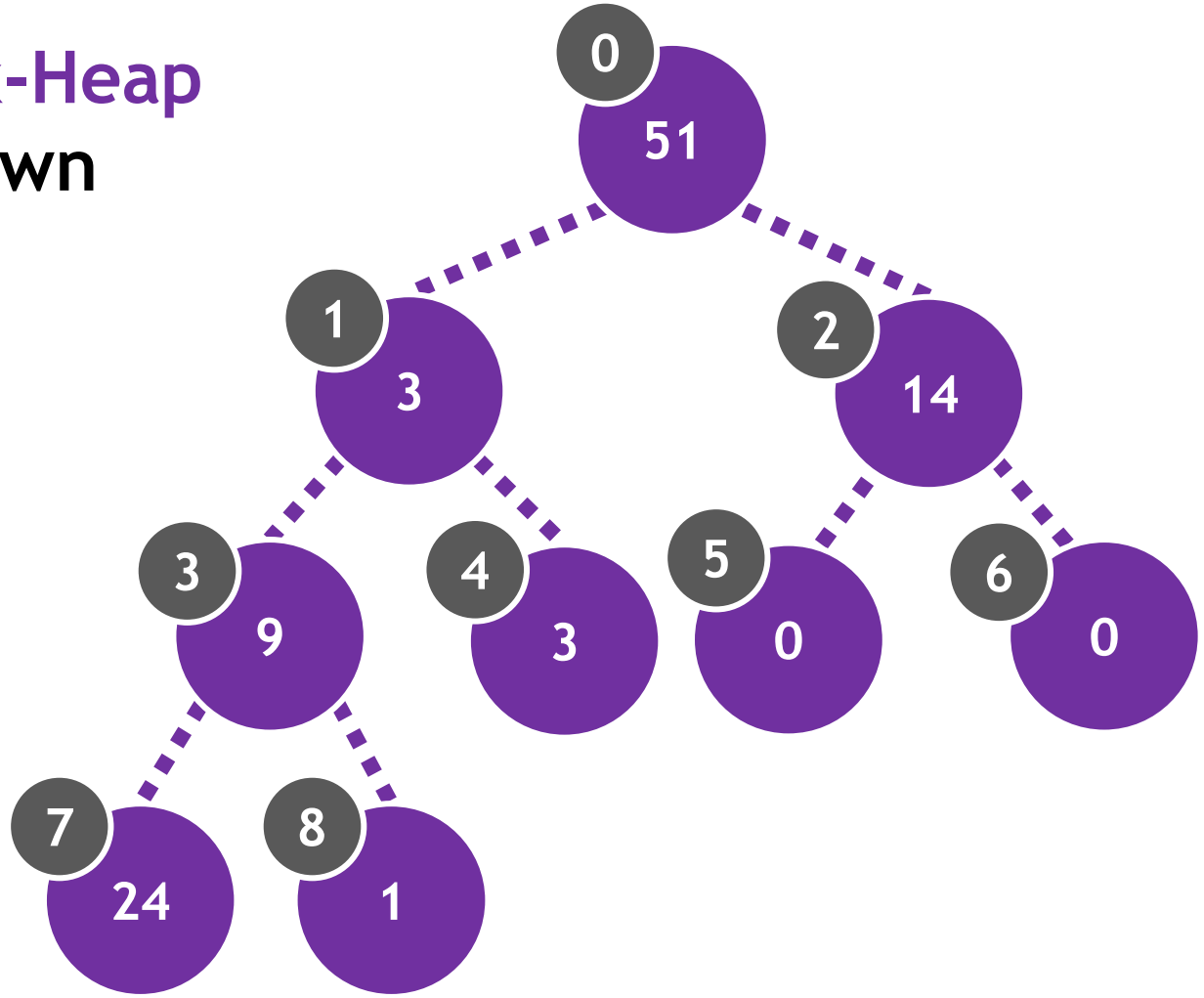


Strom, ktorý nie je v tvare Max-heap

51	3	14	9	3	0	0	24	1
0	1	2	3	4	5	6	7	8

Vytvárame Max-Heap

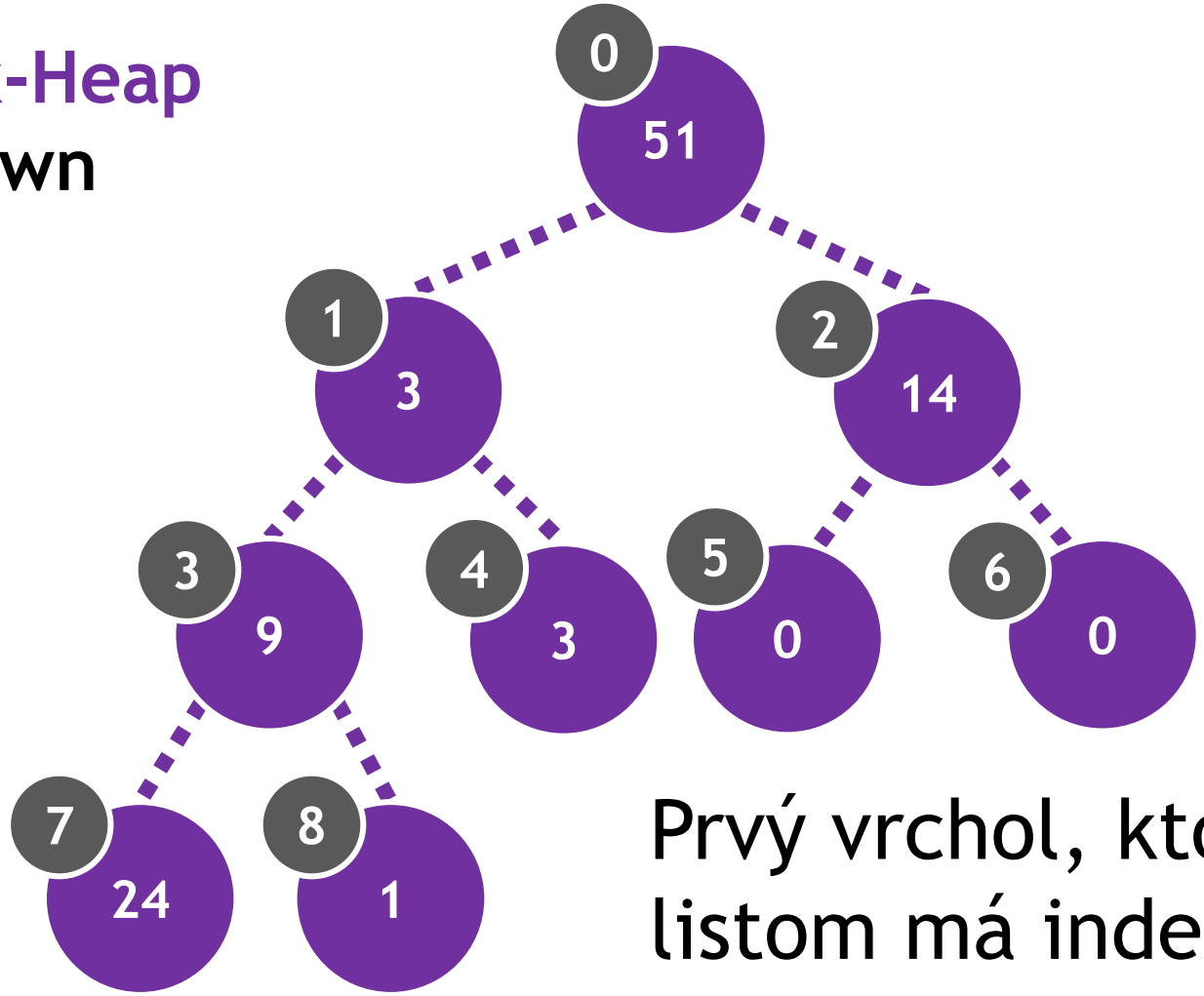
Operácia **siftDown**



51	3	14	9	3	0	0	24	1
0	1	2	3	4	5	6	7	8

Vytvárame Max-Heap

Operácia **siftDown**

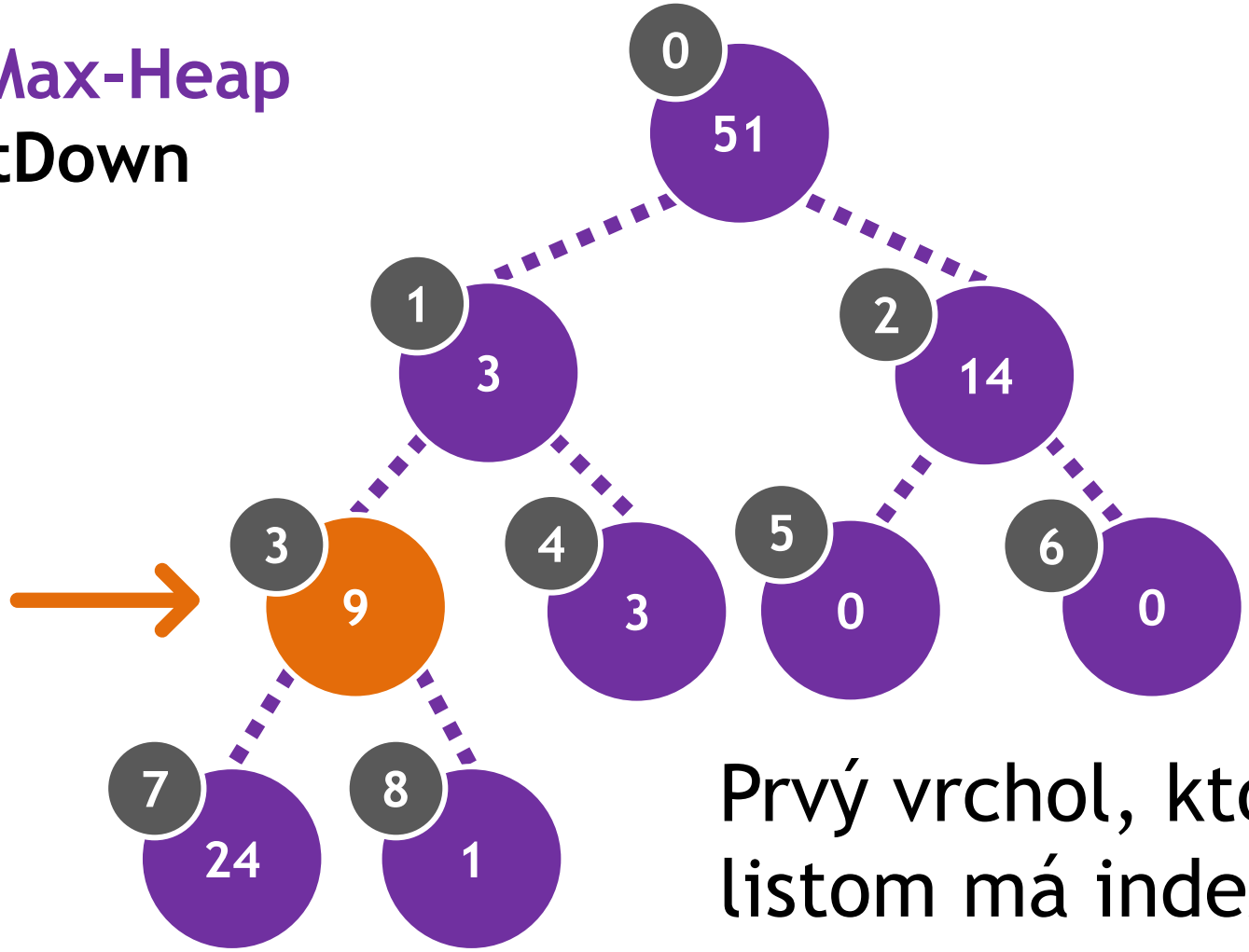


Prvý vrchol, ktorý nie je listom má index = **(n/2)-1**

51	3	14	9	3	0	0	24	1
0	1	2	3	4	5	6	7	8

Vytvárame Max-Heap

Operácia **siftDown**

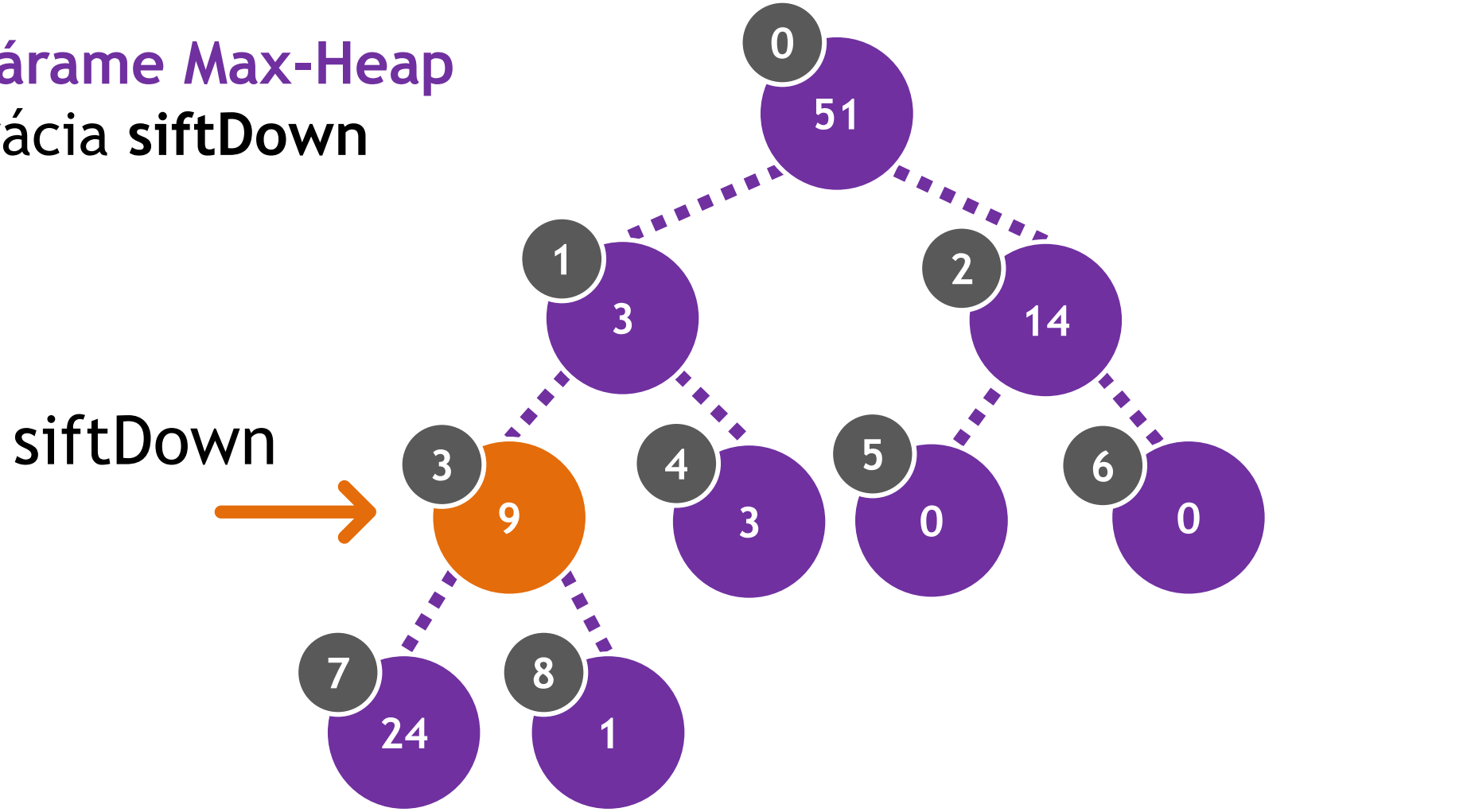


Prvý vrchol, ktorý nie je listom má index = (n/2)-1 = 3

51	3	14	9	3	0	0	24	1
0	1	2	3	4	5	6	7	8

Vytvárame Max-Heap

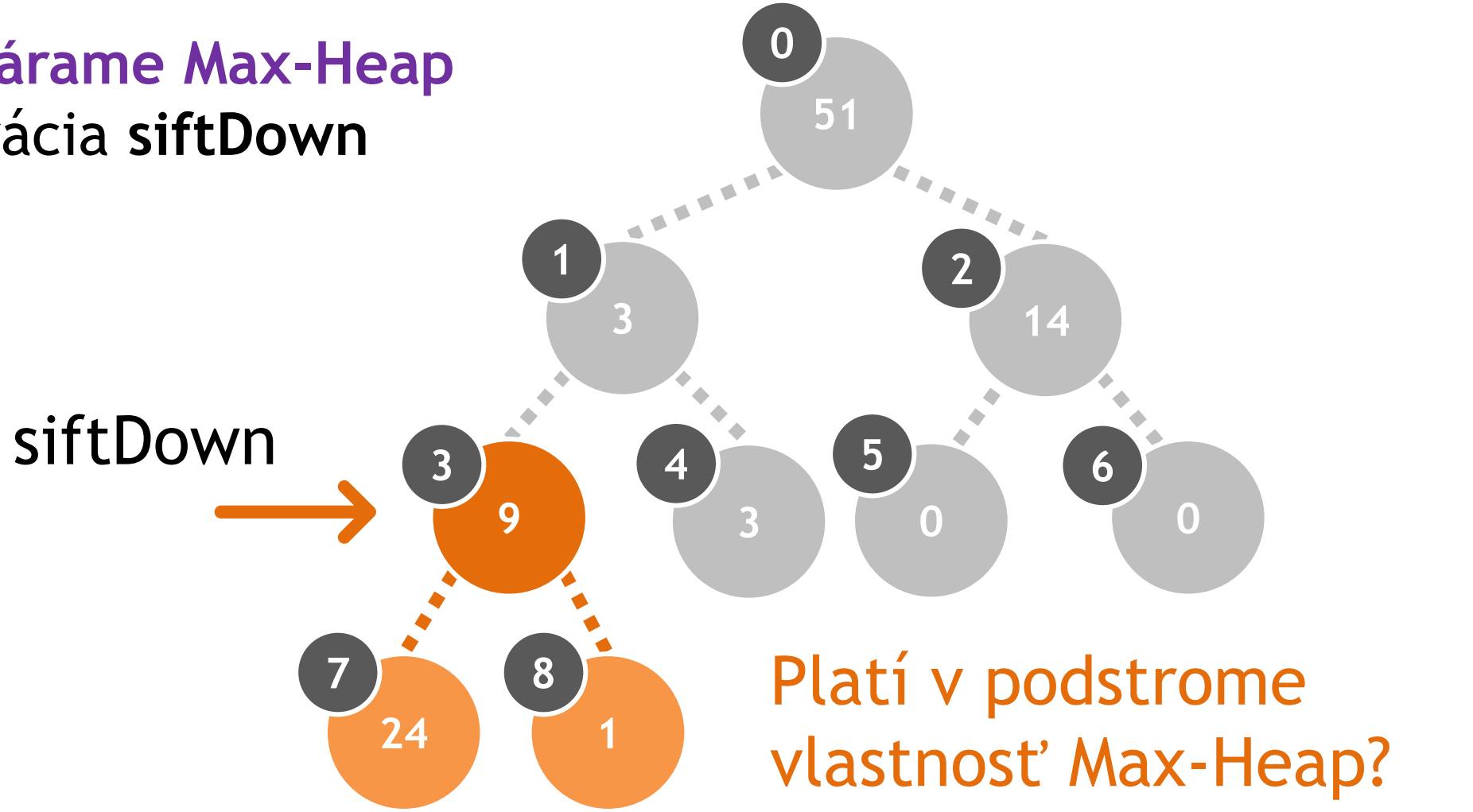
Operácia **siftDown**



51	3	14	9	3	0	0	24	1
0	1	2	3	4	5	6	7	8

Vytvárame Max-Heap

Operácia **siftDown**

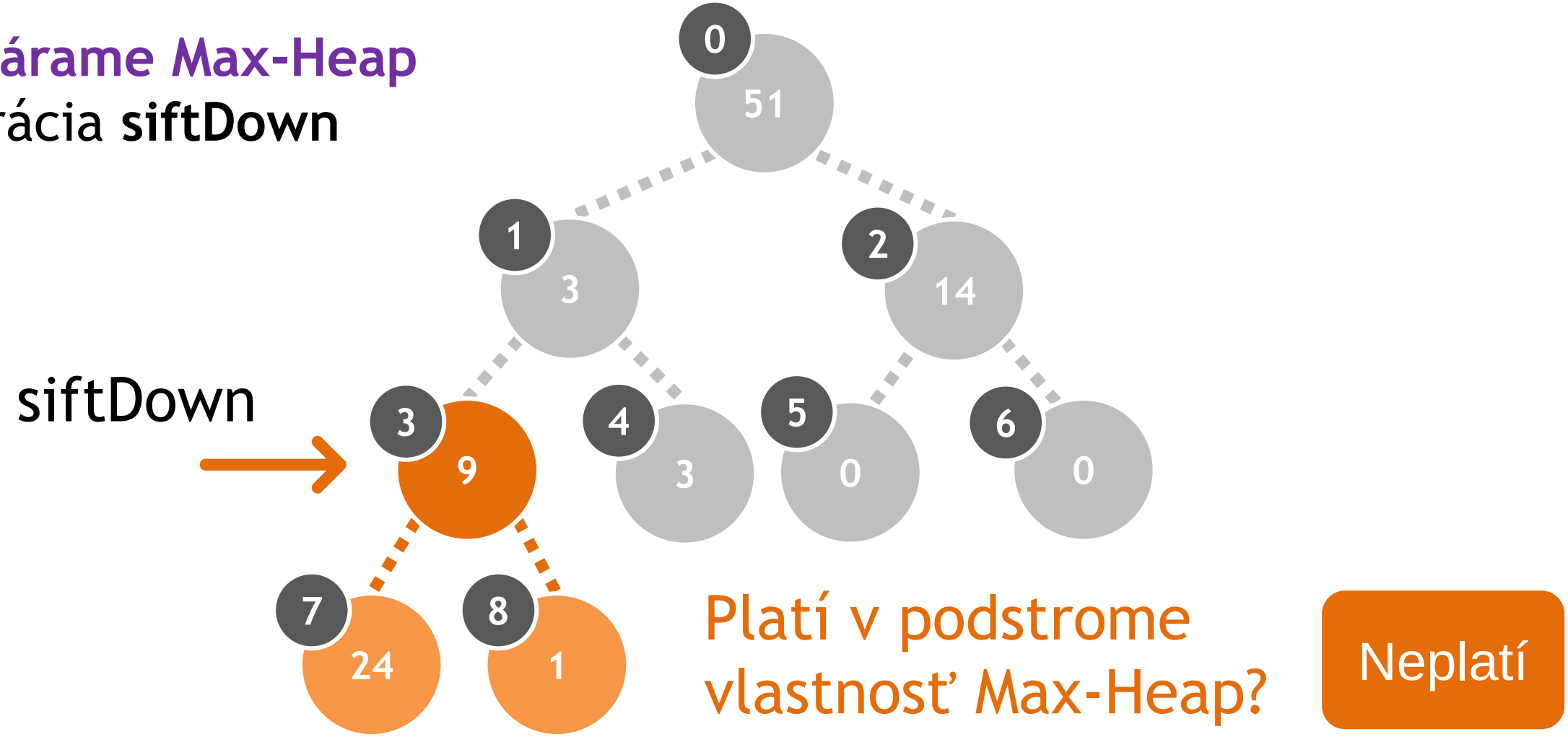


Platí v podstrome vlastnosť Max-Heap?

51	3	14	9	3	0	0	24	1
0	1	2	3	4	5	6	7	8

Vytvárame Max-Heap

Operácia **siftDown**



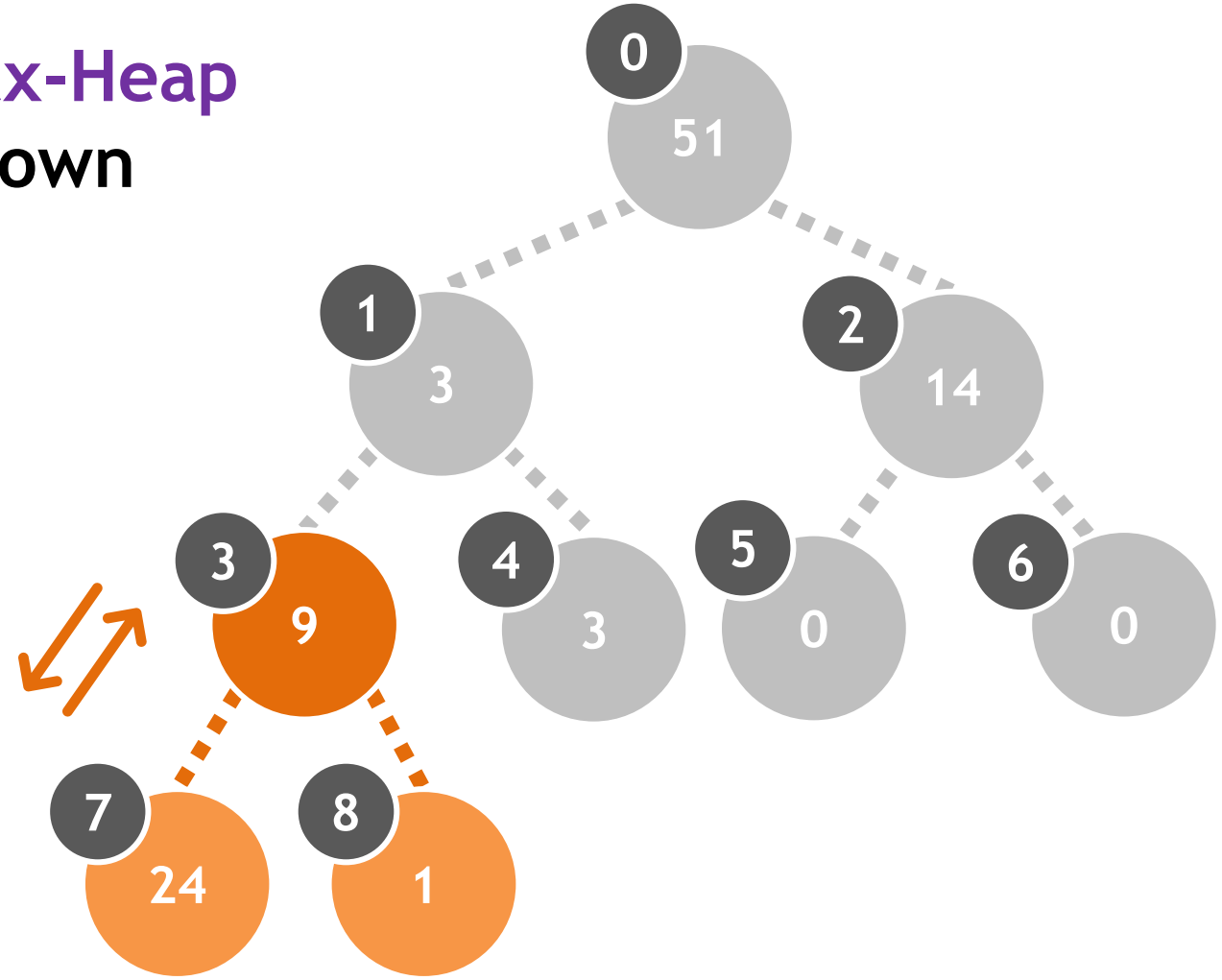
Platí v podstrome vlastnosť Max-Heap?

Neplatí

51	3	14	9	3	0	0	24	1
0	1	2	3	4	5	6	7	8

Vytvárame Max-Heap

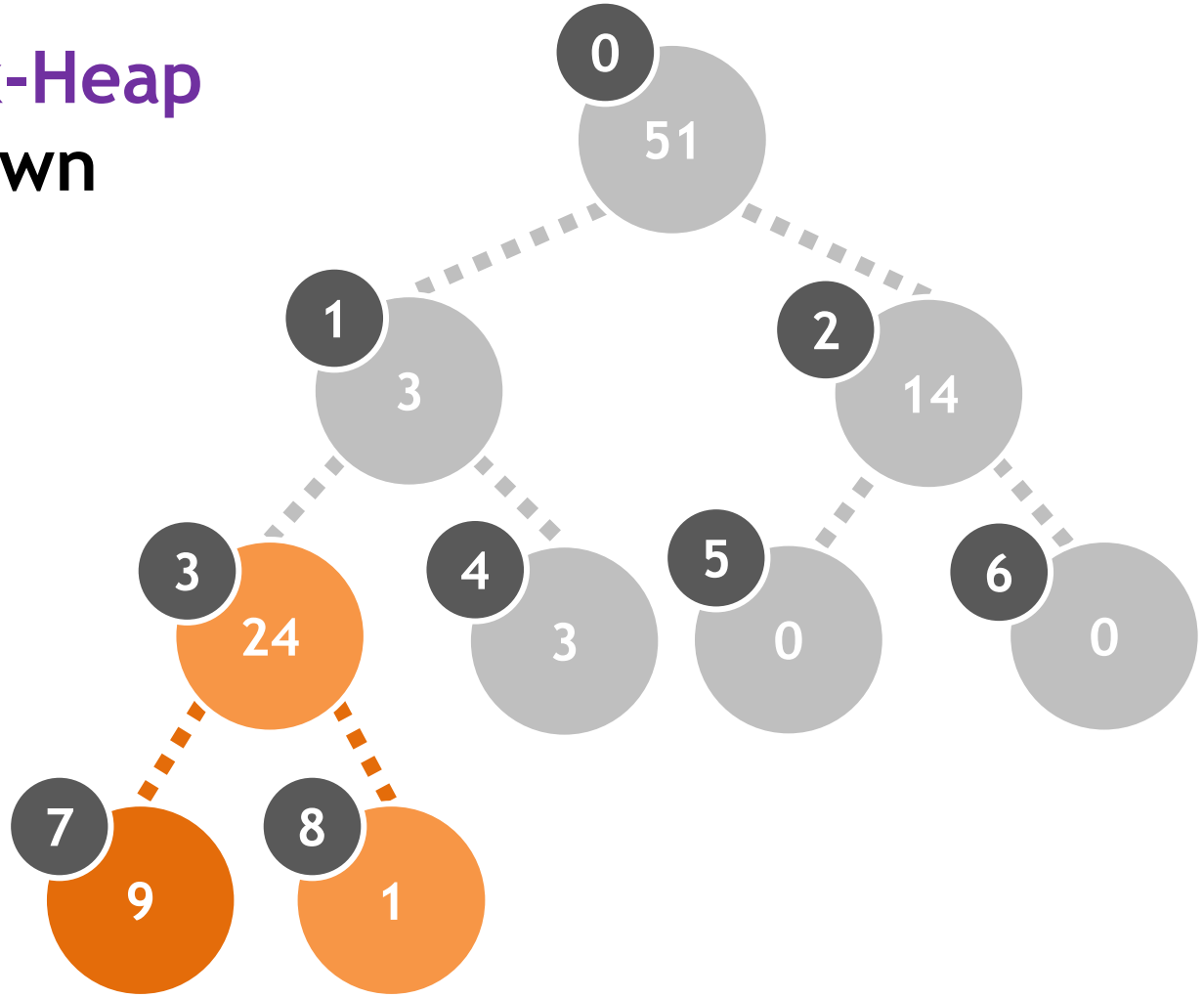
Operácia **siftDown**



51	3	14	9	3	0	0	24	1
0	1	2	3	4	5	6	7	8

Vytvárame Max-Heap

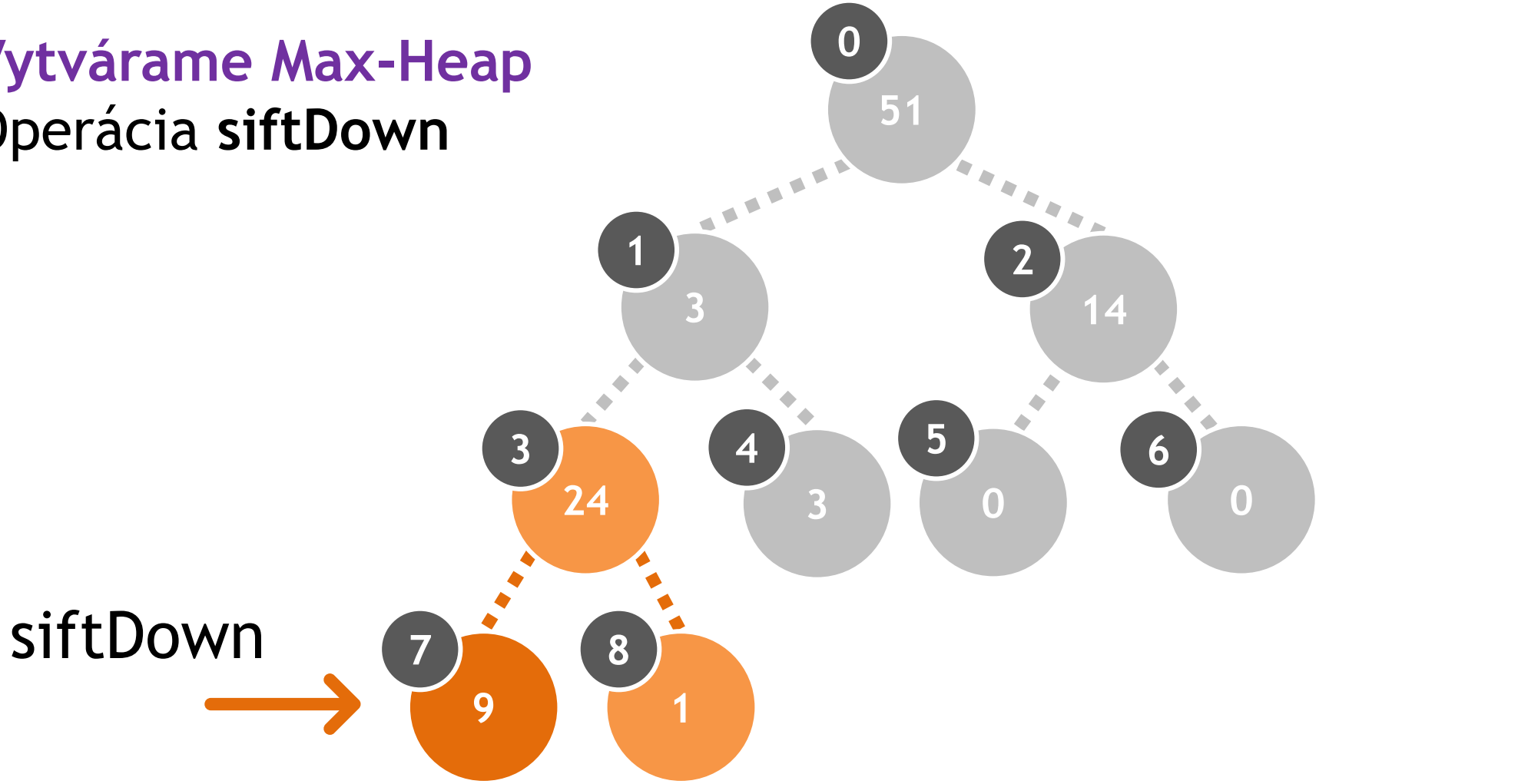
Operácia **siftDown**



51	3	14	24	3	0	0	9	1
0	1	2	3	4	5	6	7	8

Vytvárame Max-Heap

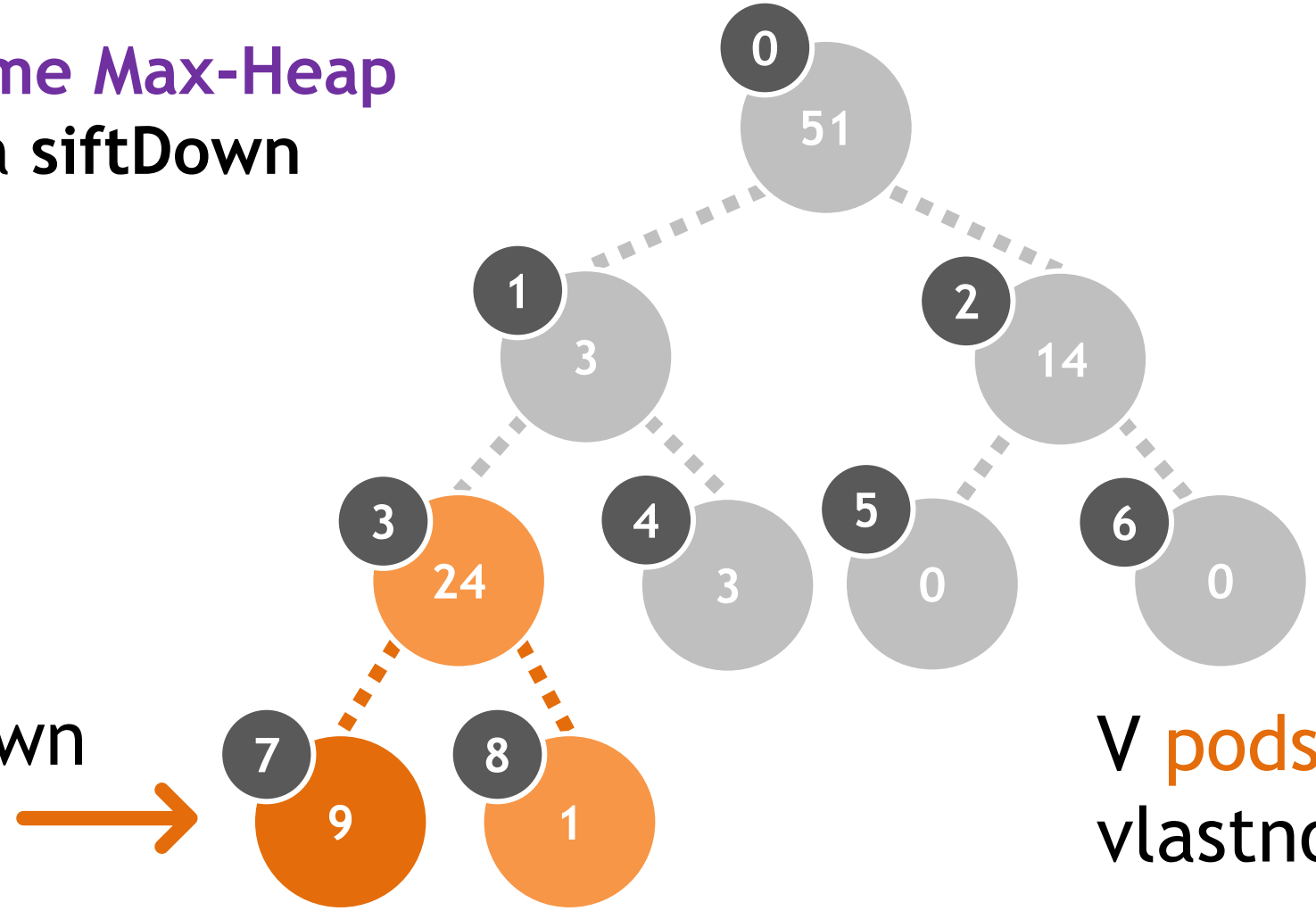
Operácia **siftDown**



51	3	14	24	3	0	0	9	1
0	1	2	3	4	5	6	7	8

Vytvárame Max-Heap

Operácia **siftDown**

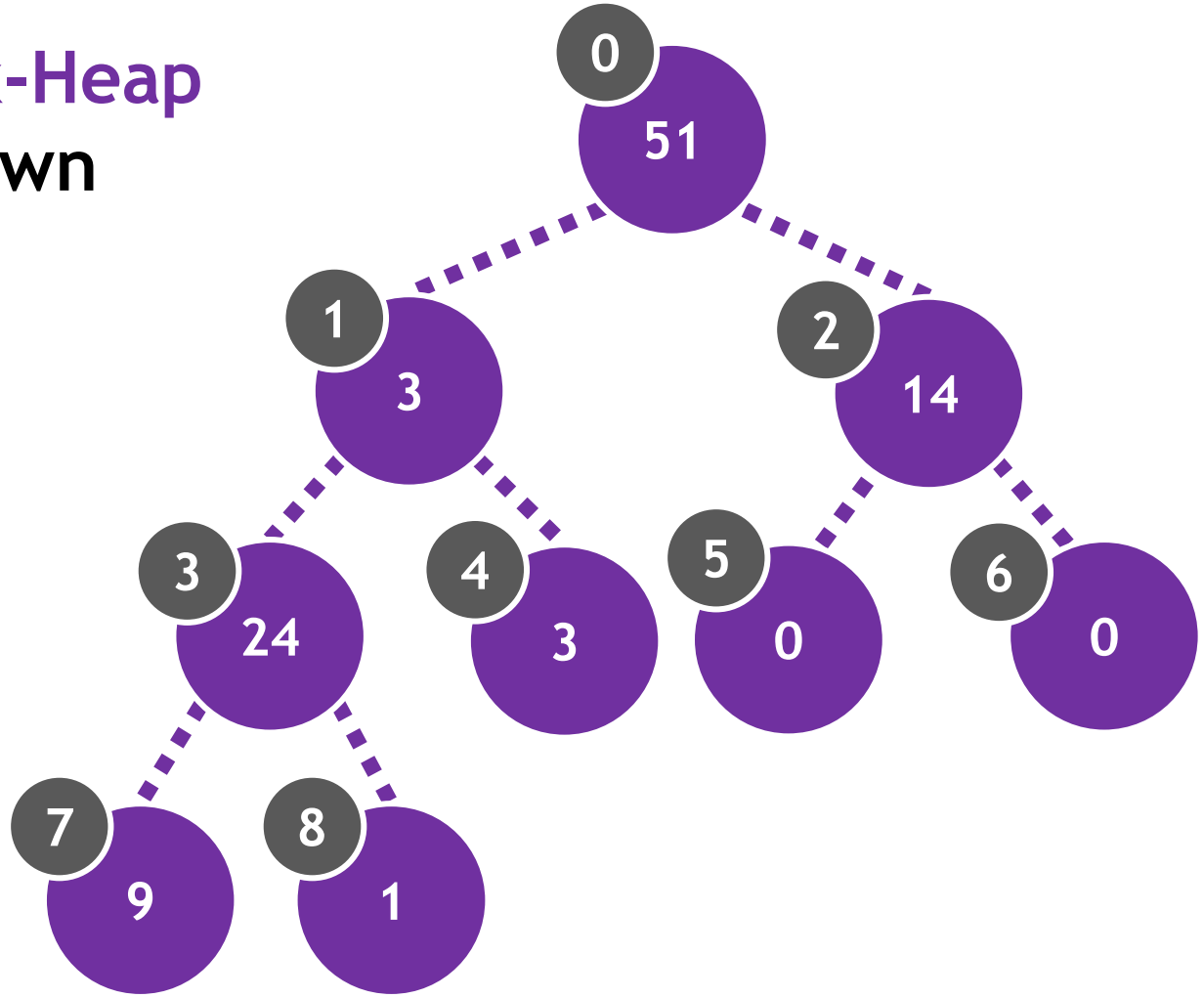


V podstrome platí vlastnosť Max-Heap.

51	3	14	24	3	0	0	9	1
0	1	2	3	4	5	6	7	8

Vytvárame Max-Heap

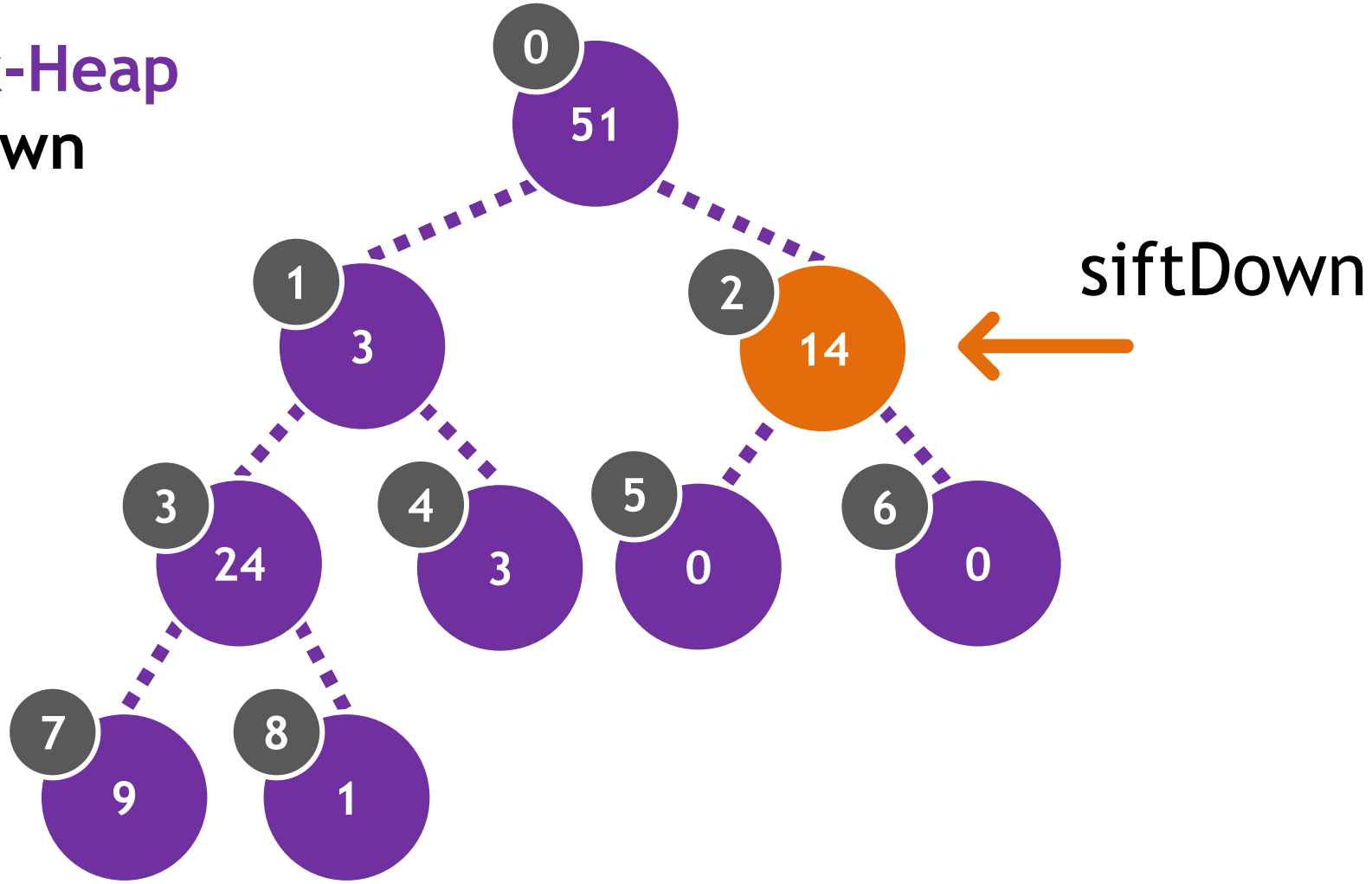
Operácia **siftDown**



51	3	14	24	3	0	0	9	1
0	1	2	3	4	5	6	7	8

Vytvárame Max-Heap

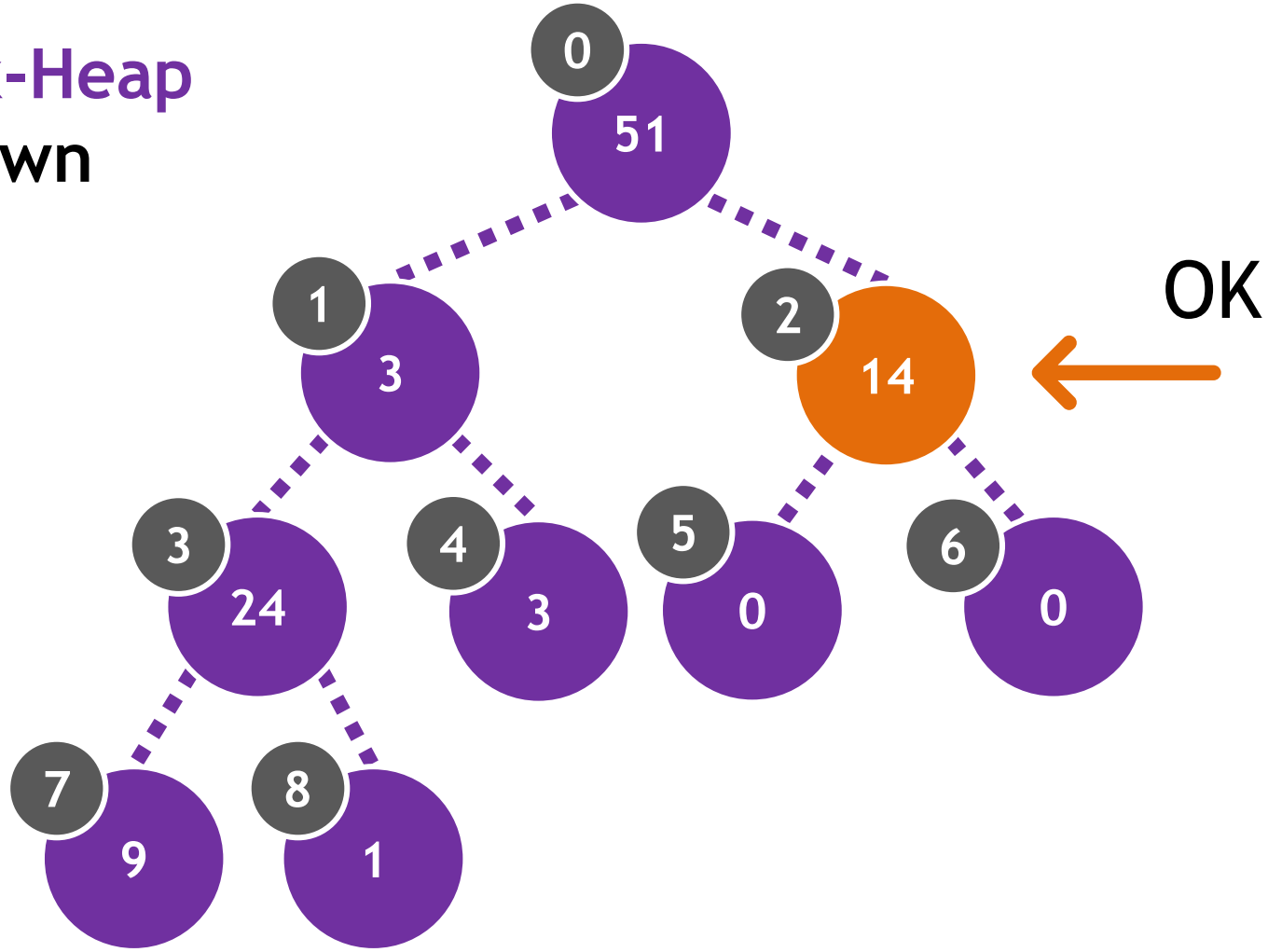
Operácia **siftDown**



51	3	14	24	3	0	0	9	1
0	1	2	3	4	5	6	7	8

Vytvárame Max-Heap

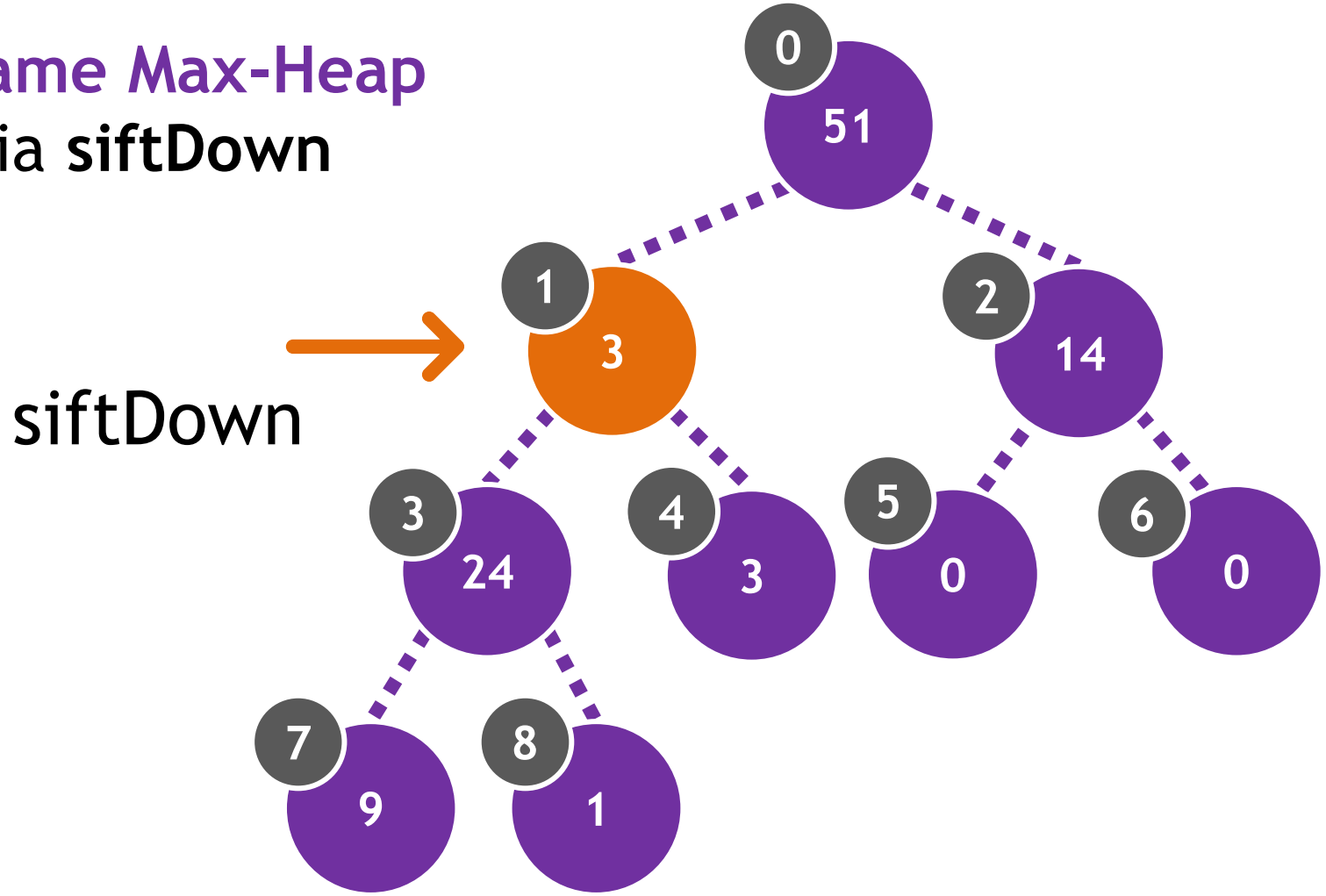
Operácia **siftDown**



51	3	14	24	3	0	0	9	1
0	1	2	3	4	5	6	7	8

Vytvárame Max-Heap

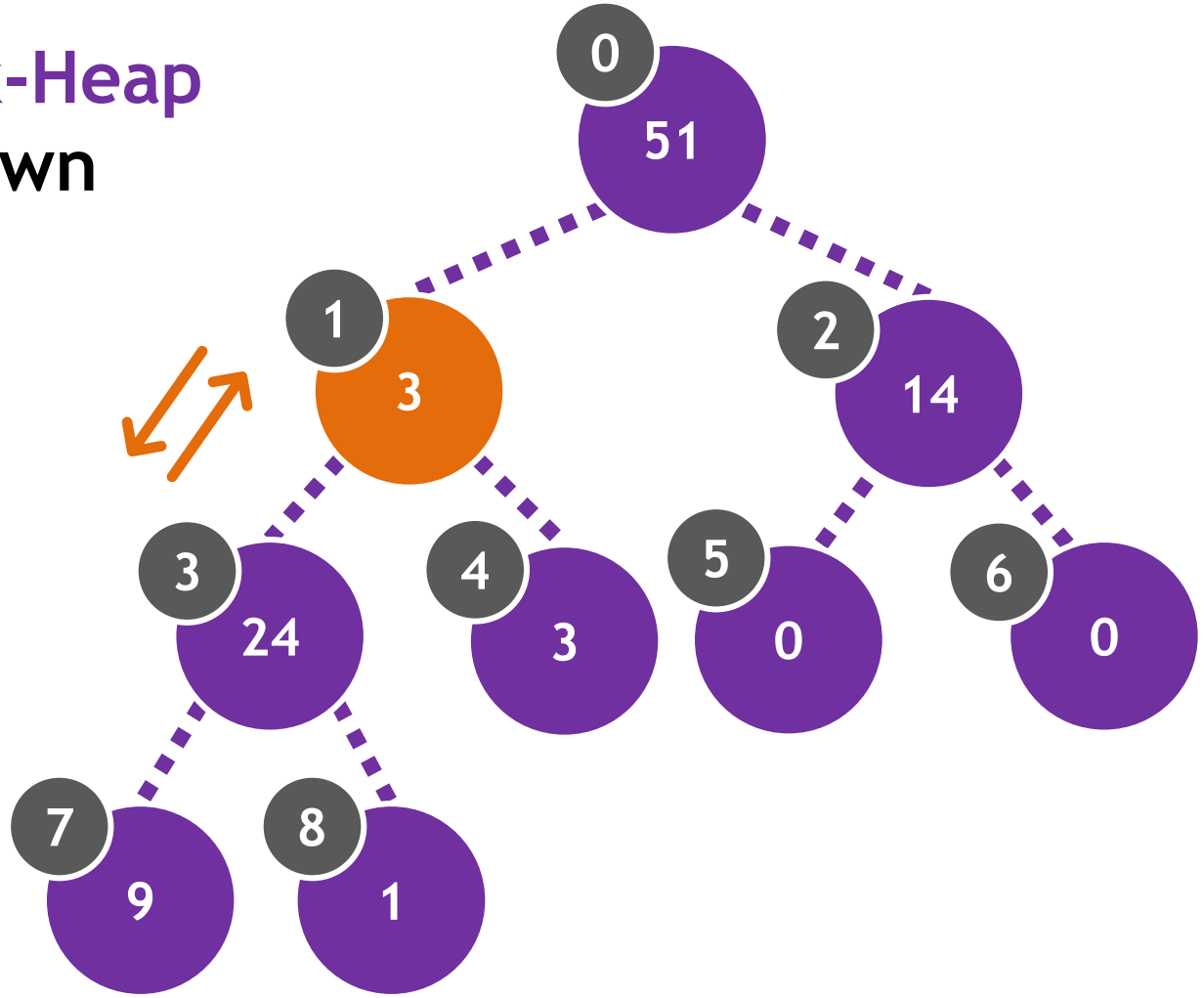
Operácia **siftDown**



51	3	14	24	3	0	0	9	1
0	1	2	3	4	5	6	7	8

Vytvárame Max-Heap

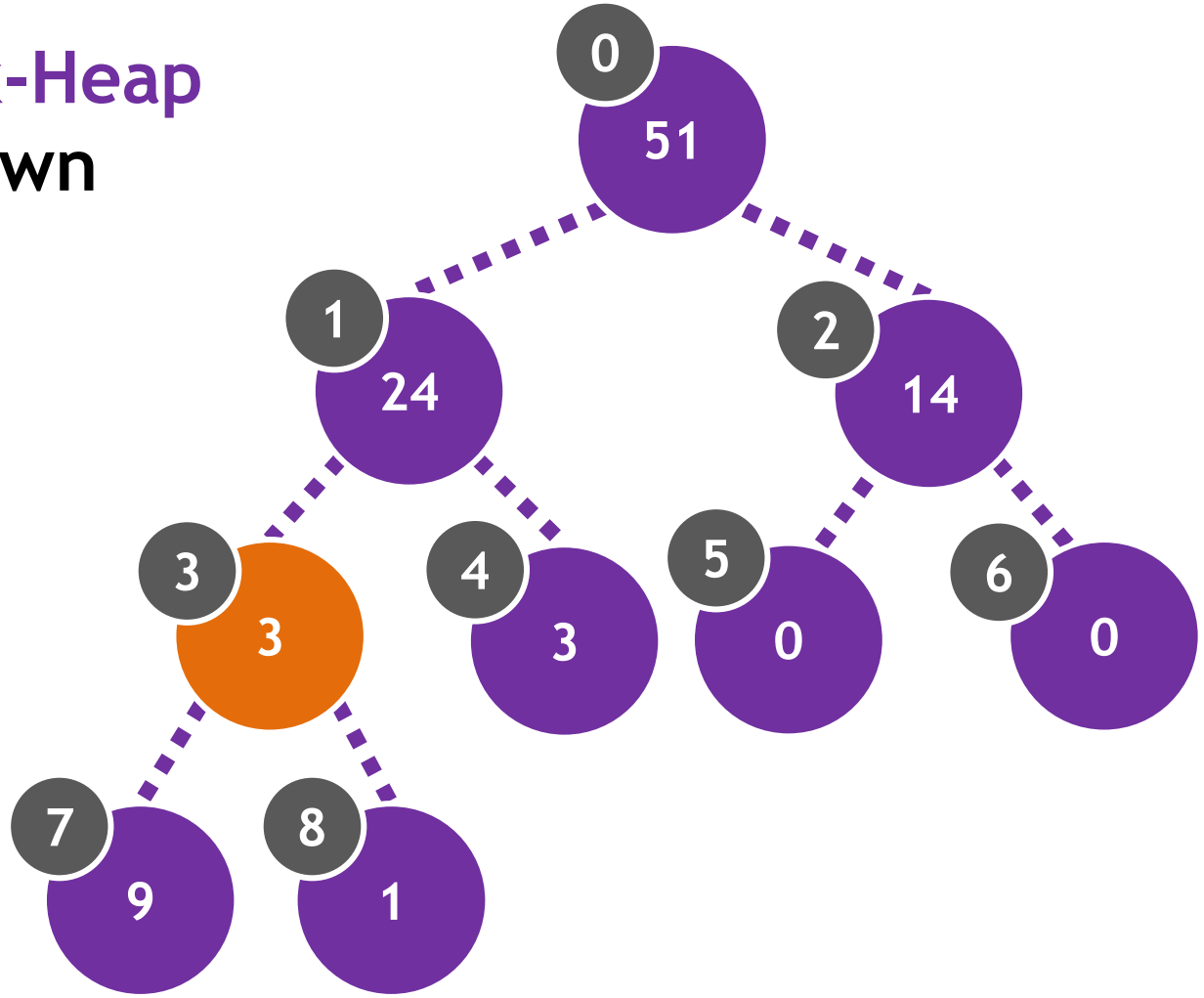
Operácia **siftDown**



51	3	14	24	3	0	0	9	1
0	1	2	3	4	5	6	7	8

Vytvárame Max-Heap

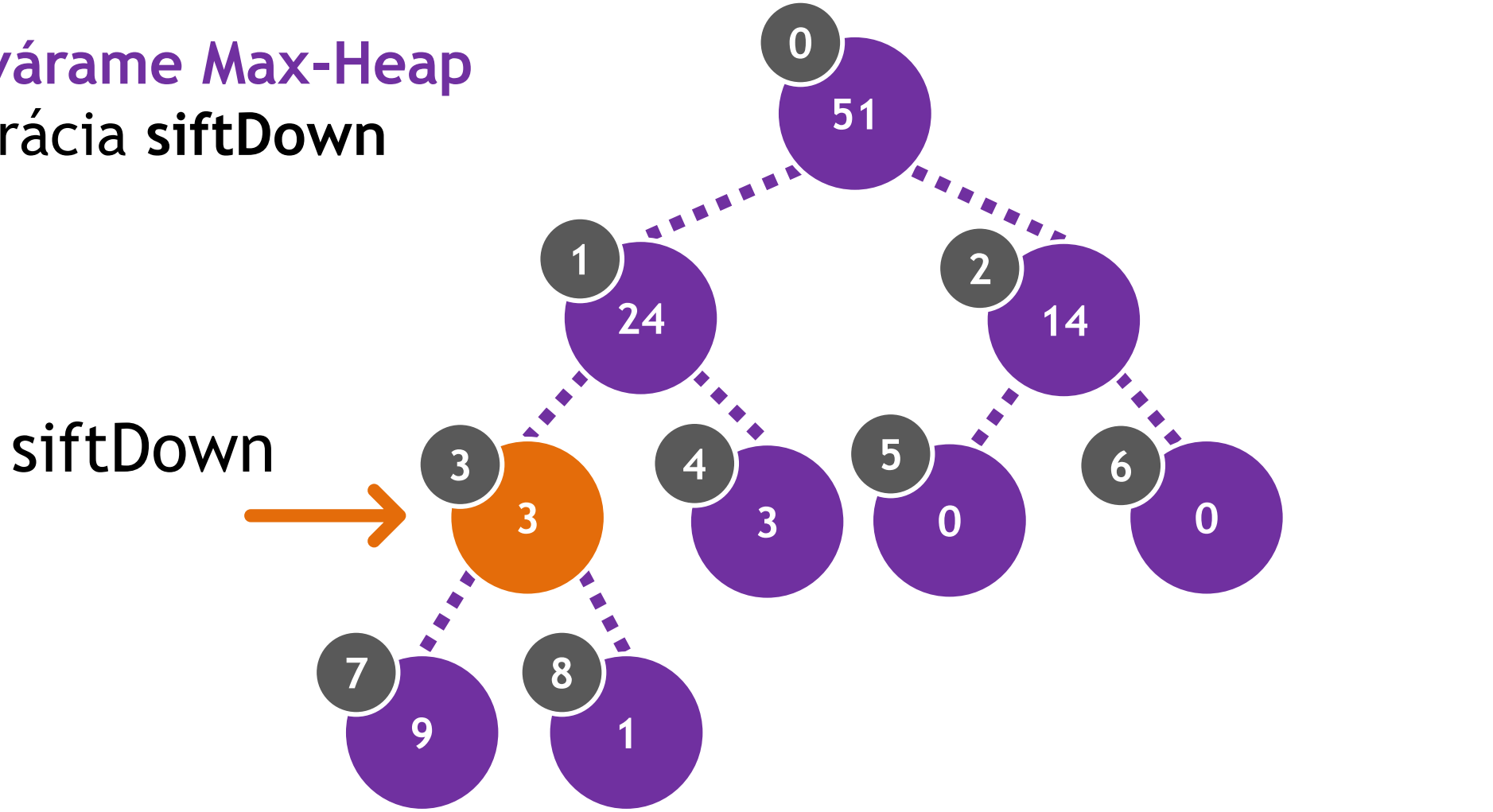
Operácia **siftDown**



51	24	14	3	3	0	0	9	1
0	1	2	3	4	5	6	7	8

Vytvárame Max-Heap

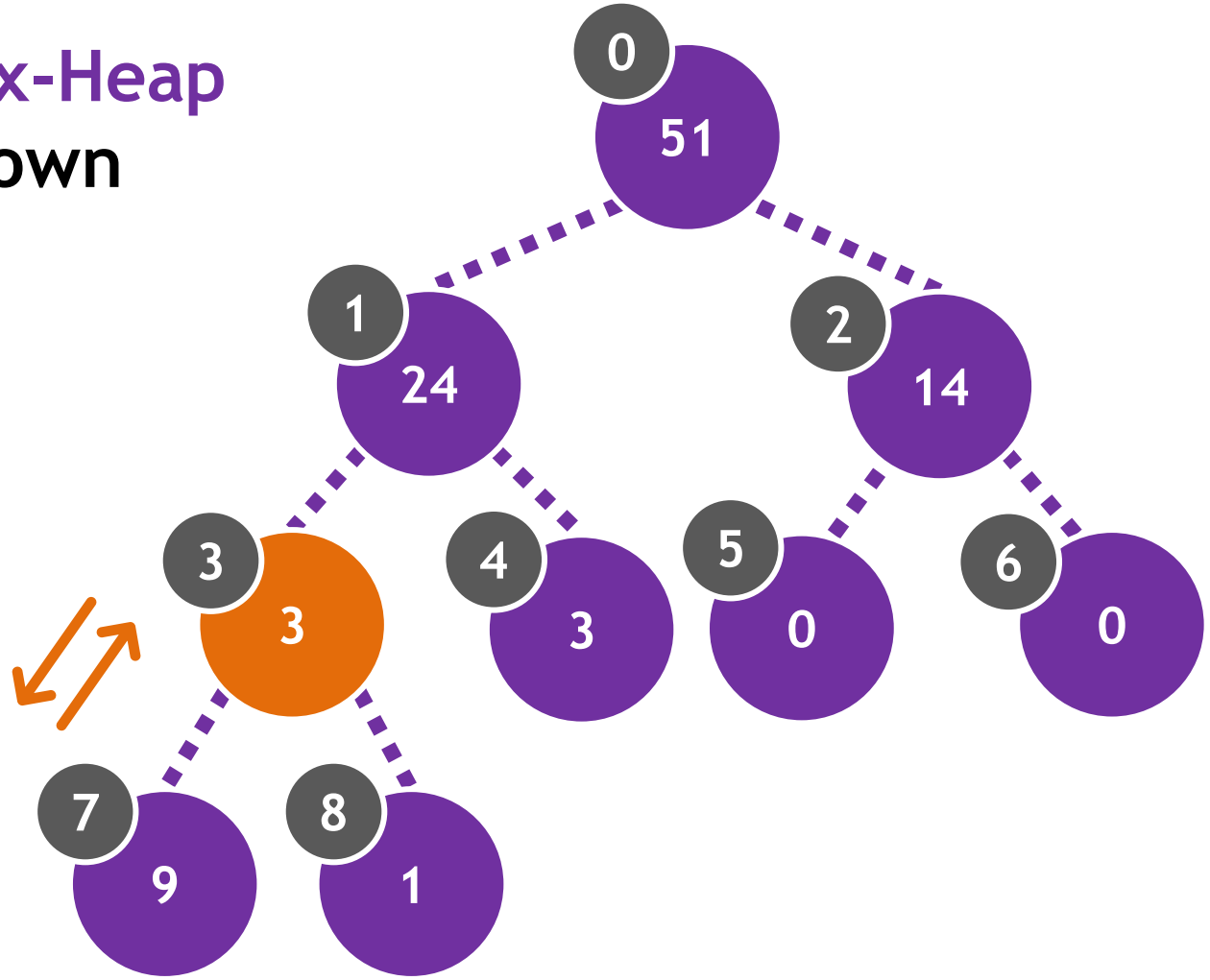
Operácia **siftDown**



51	24	14	3	3	0	0	9	1
0	1	2	3	4	5	6	7	8

Vytvárame Max-Heap

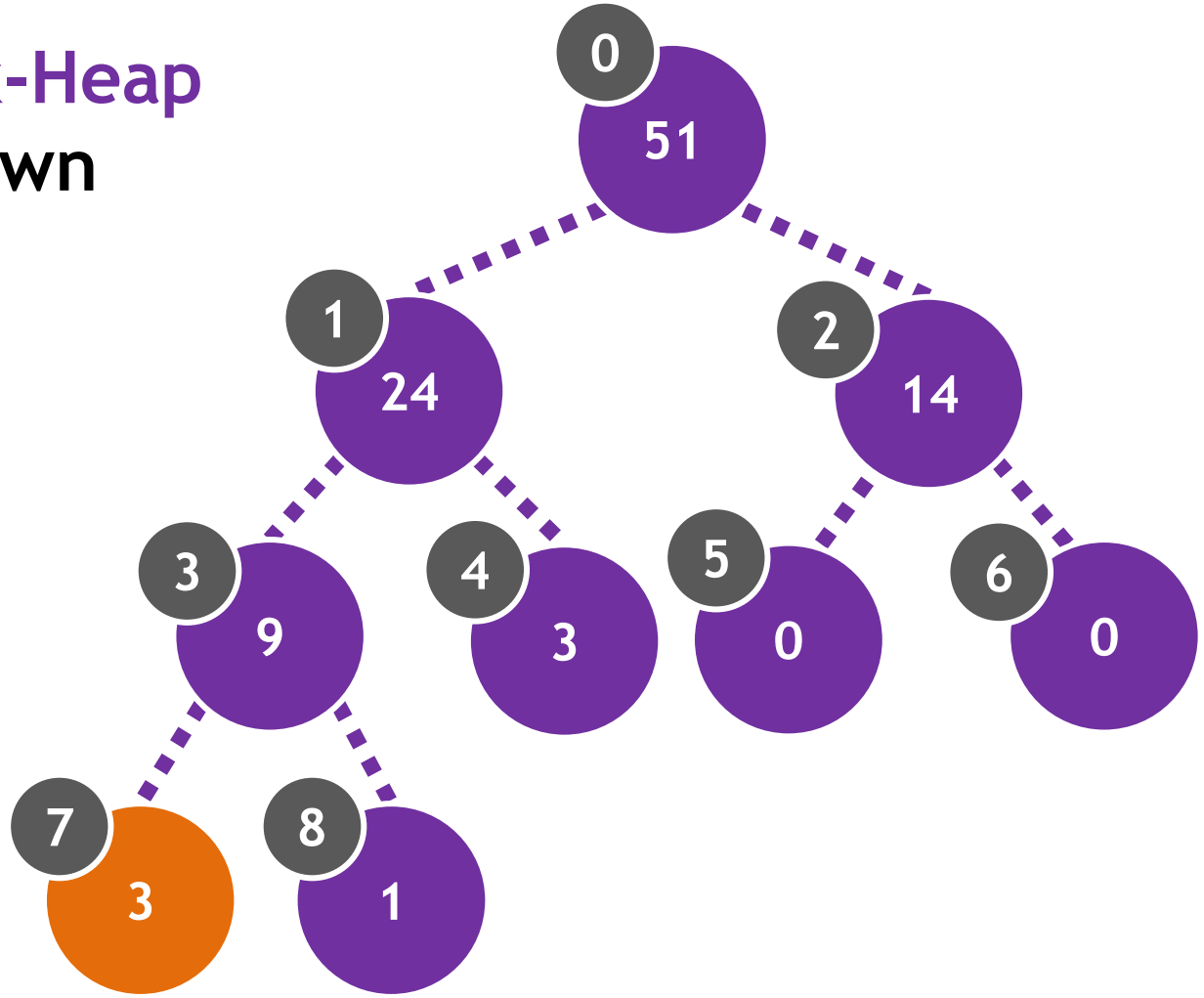
Operácia **siftDown**



51	24	14	3	3	0	0	9	1
0	1	2	3	4	5	6	7	8

Vytvárame Max-Heap

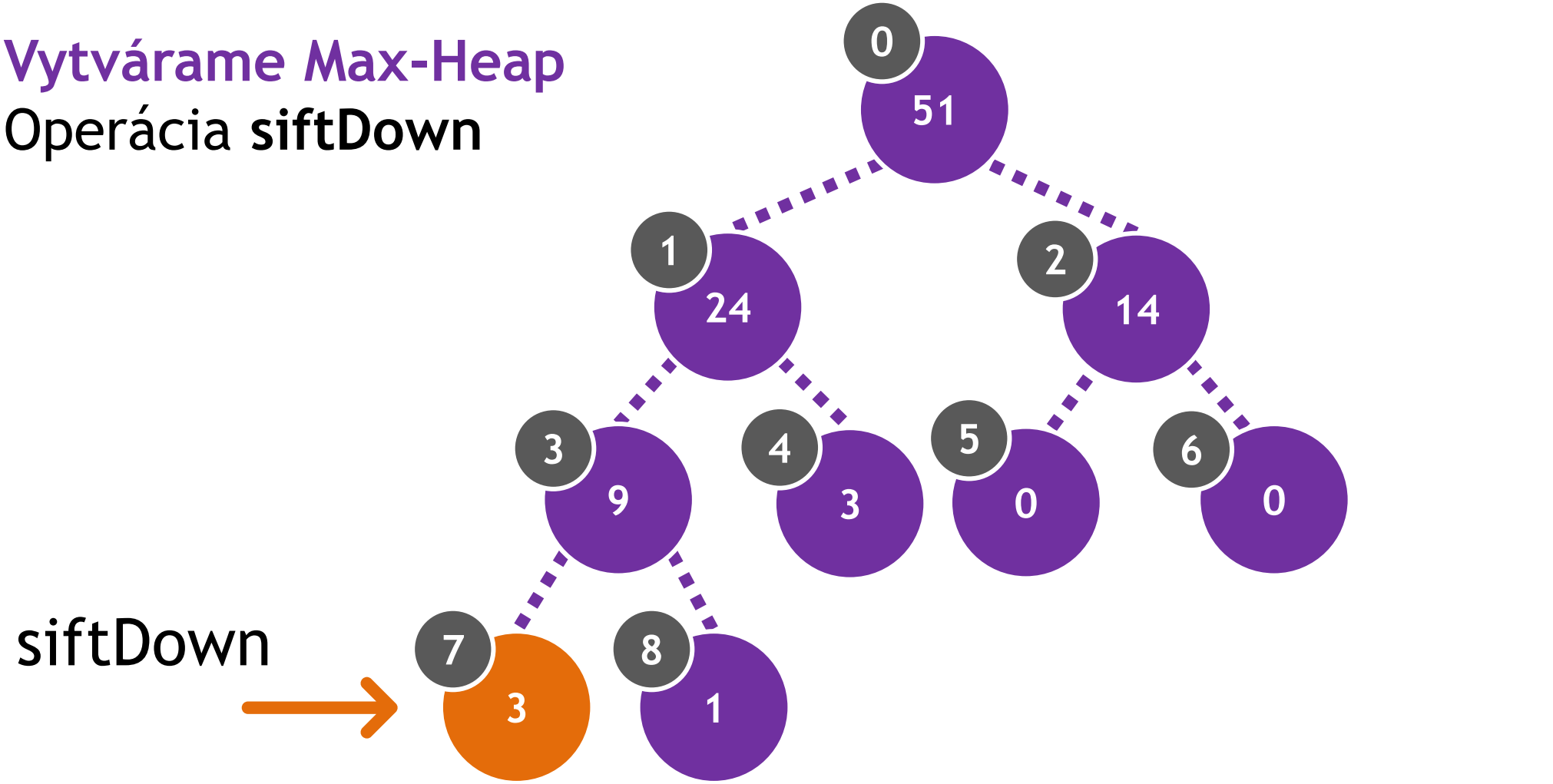
Operácia **siftDown**



51	24	14	9	3	0	0	3	1
0	1	2	3	4	5	6	7	8

Vytvárame Max-Heap

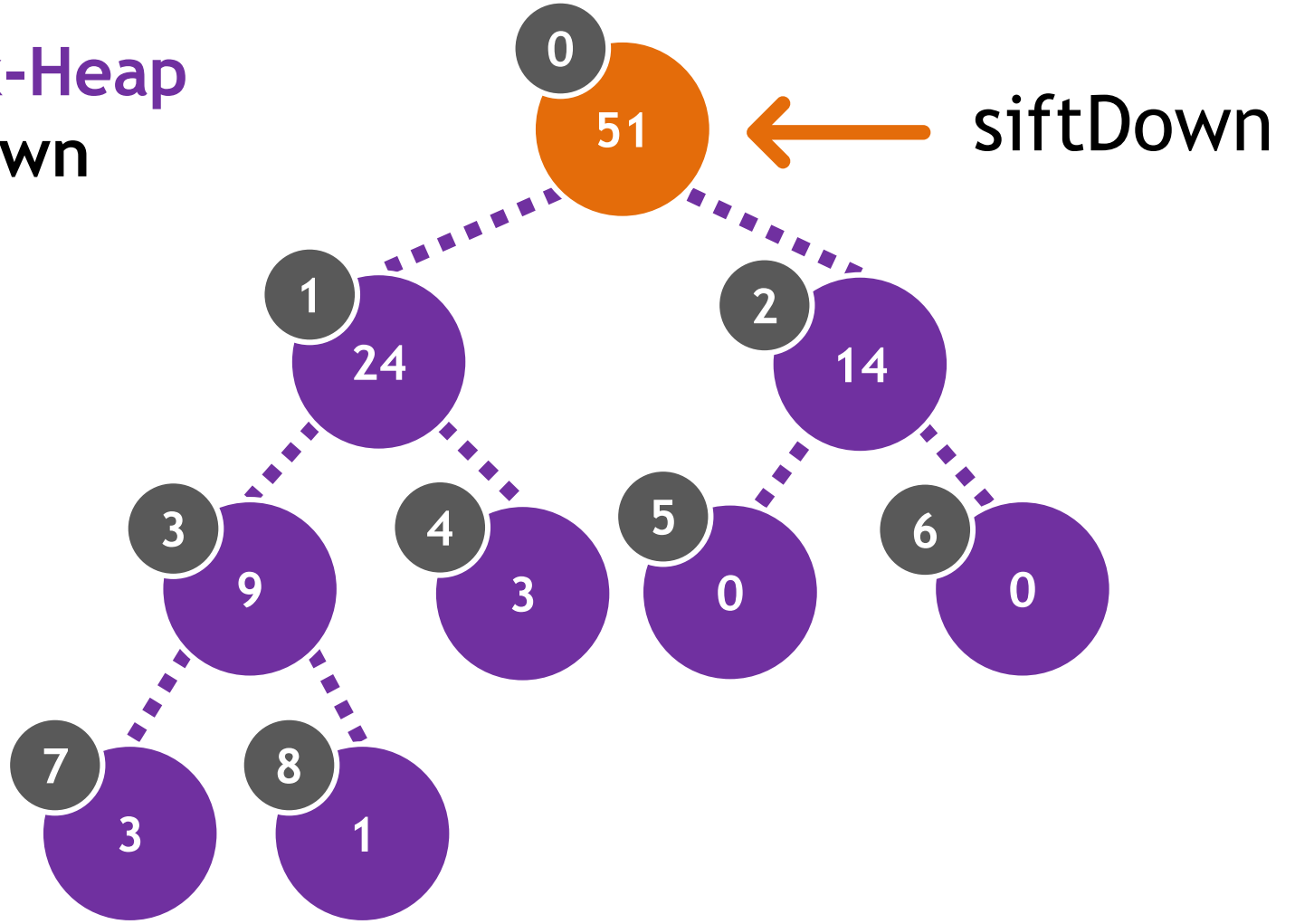
Operácia **siftDown**



51	24	14	9	3	0	0	3	1
0	1	2	3	4	5	6	7	8

Vytvárame Max-Heap

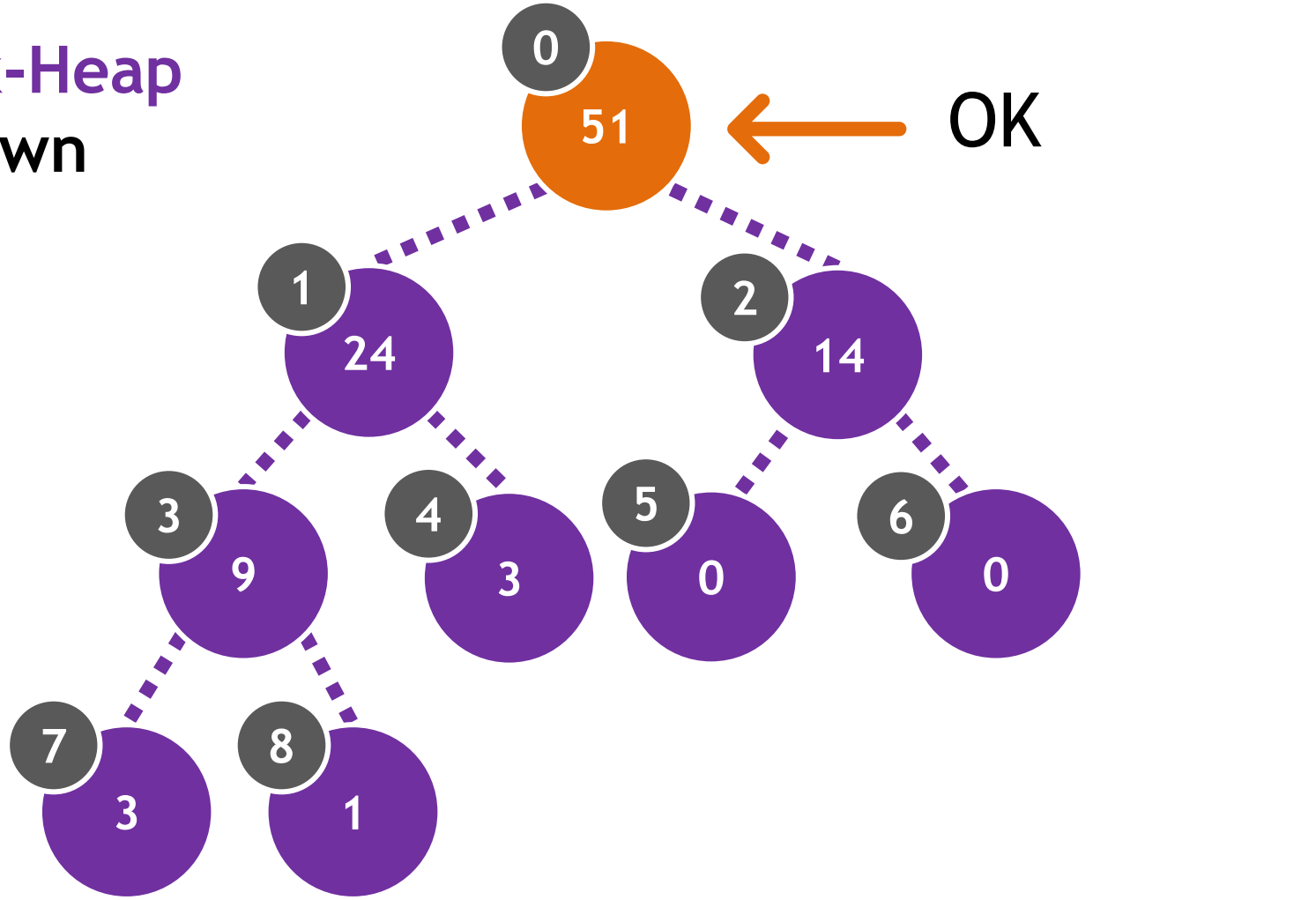
Operácia **siftDown**



51	24	14	9	3	0	0	3	1
0	1	2	3	4	5	6	7	8

Vytvárame Max-Heap

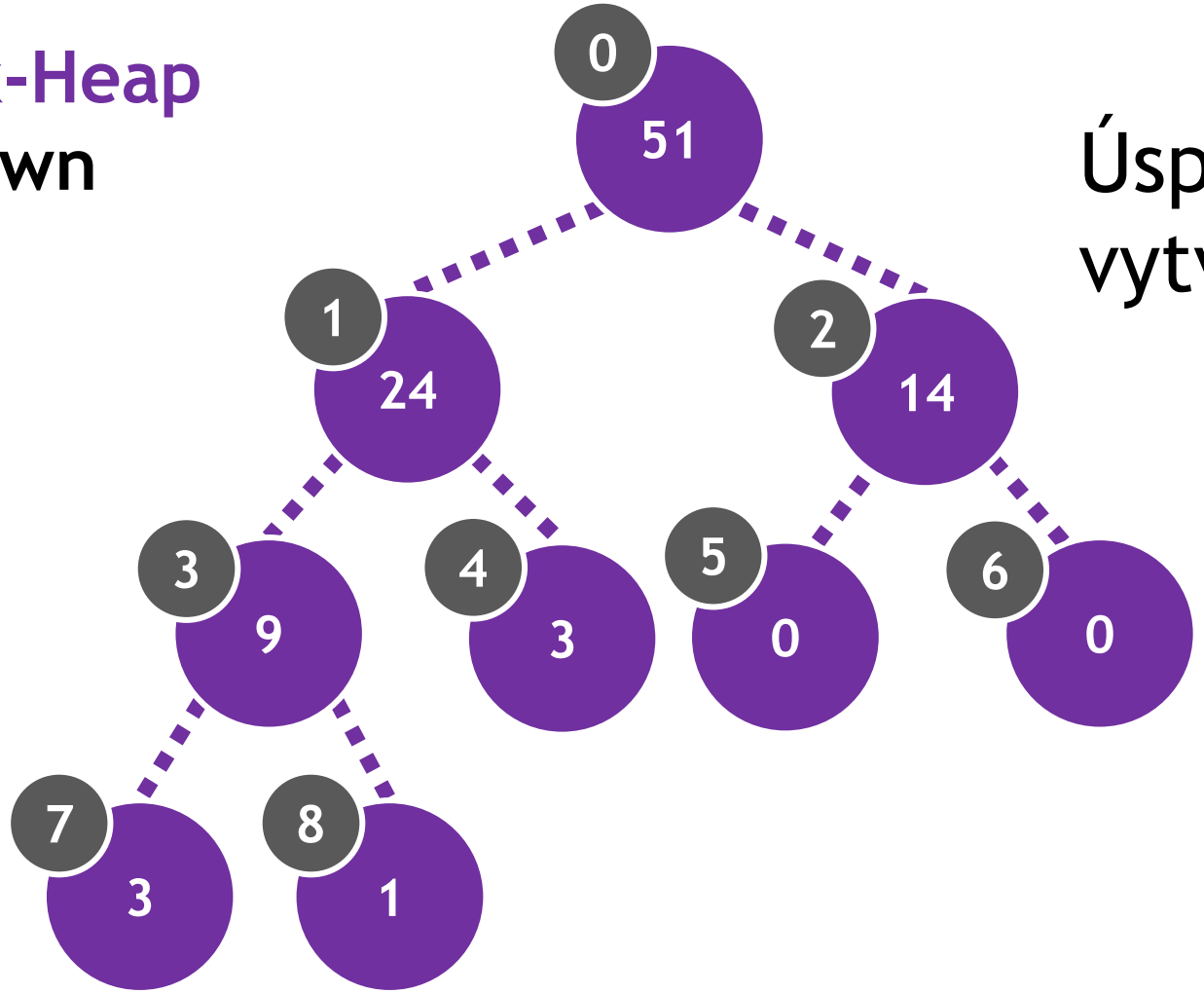
Operácia **siftDown**



51	24	14	9	3	0	0	3	1
0	1	2	3	4	5	6	7	8

Vytvárame Max-Heap

Operácia **siftDown**

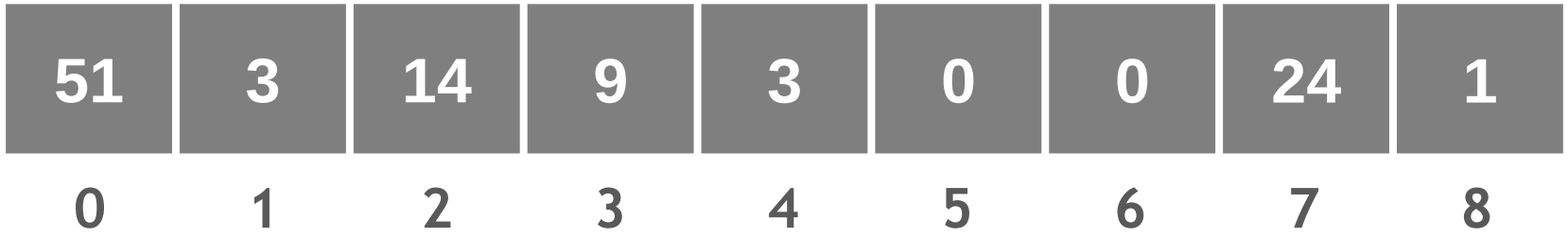


Úspešne sme vytvorili **Max-Heap**

51	24	14	9	3	0	0	3	1
0	1	2	3	4	5	6	7	8

Vytvárame Max-Heap

Pomocou operácie **siftUp**



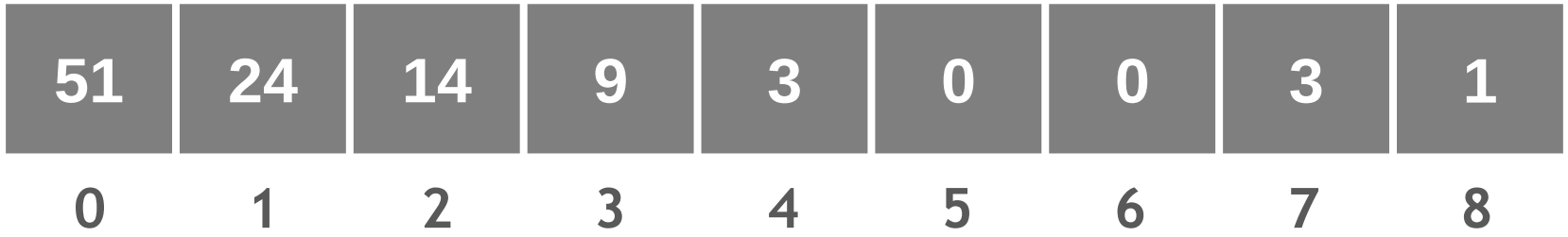
Vytvárame Max-Heap

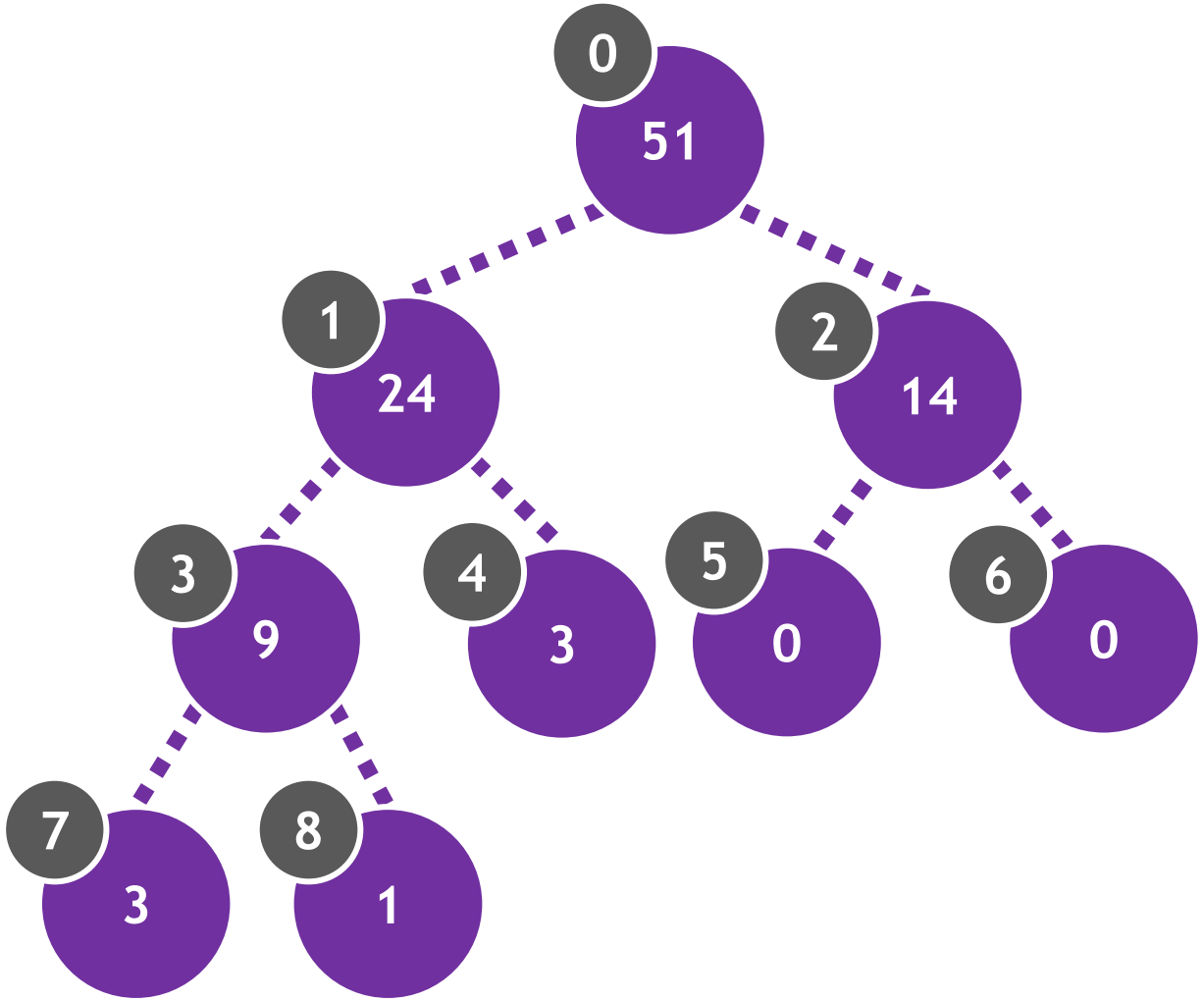
Pomocou operácie **siftUp**

Samoštúdium

51	3	14	9	3	0	0	24	1
0	1	2	3	4	5	6	7	8

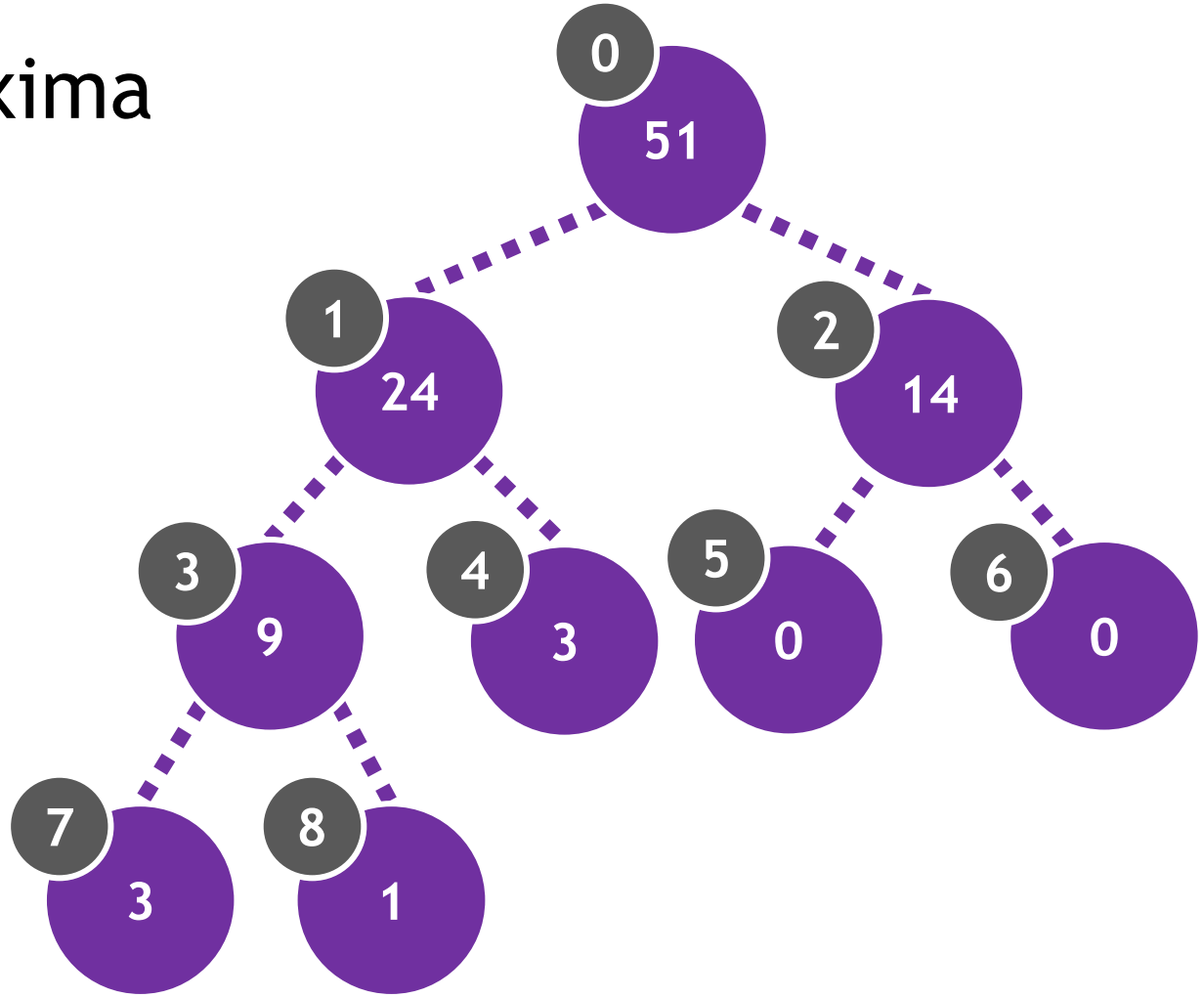
2. Fáza Triedenie





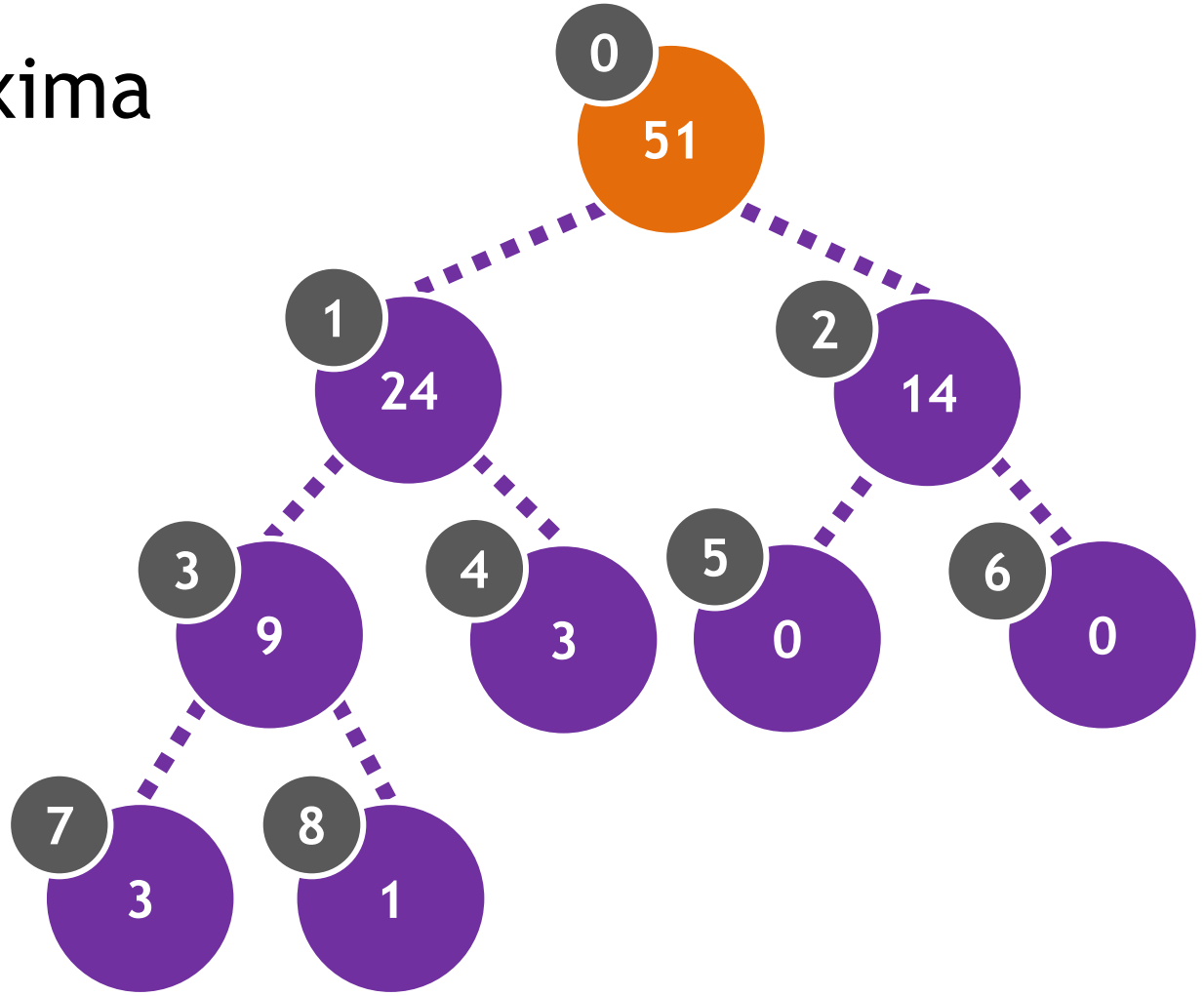
51	24	14	9	3	0	0	3	1
0	1	2	3	4	5	6	7	8

Extrakcia maxima



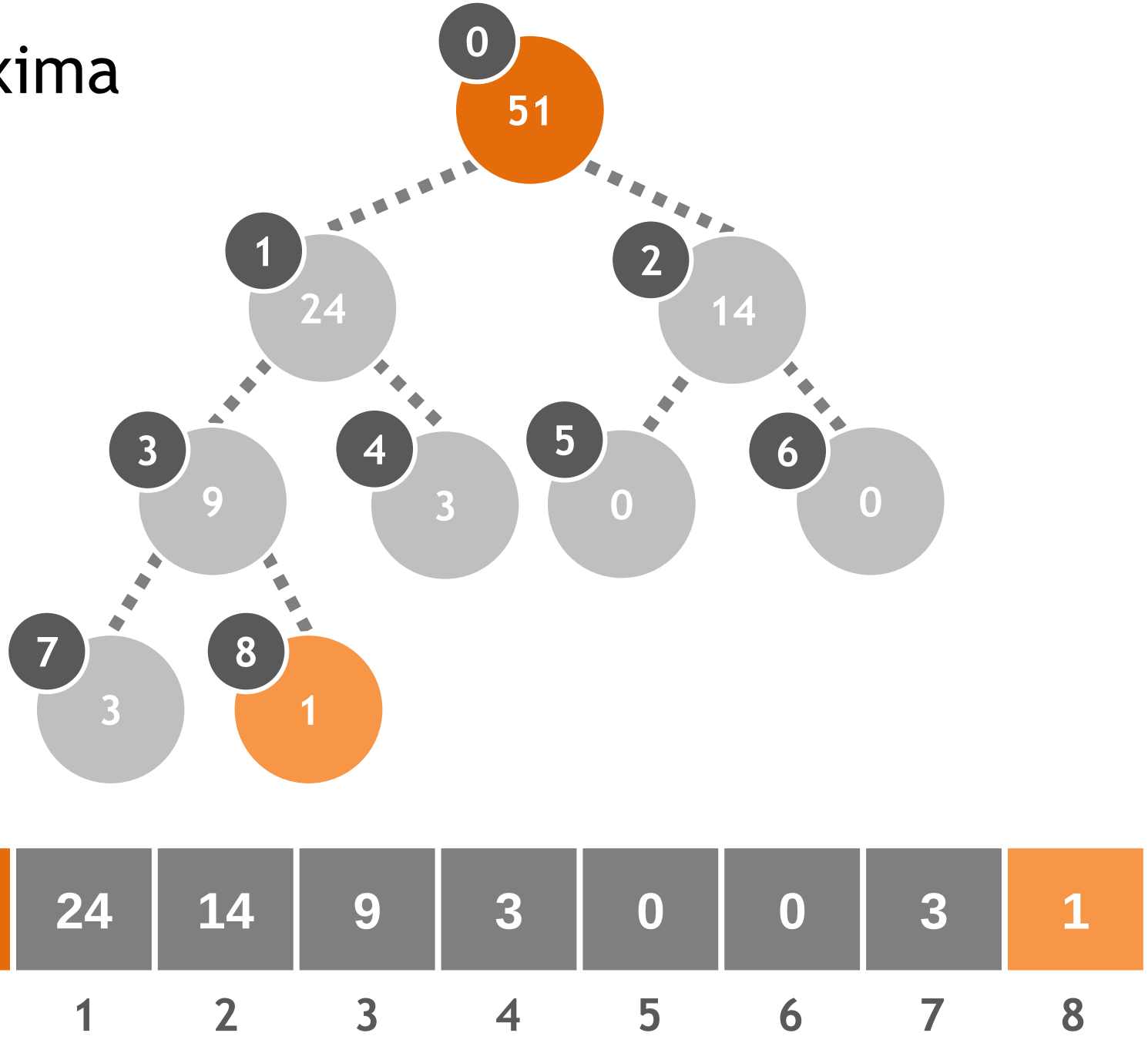
51	24	14	9	3	0	0	3	1
0	1	2	3	4	5	6	7	8

Extrakcia maxima

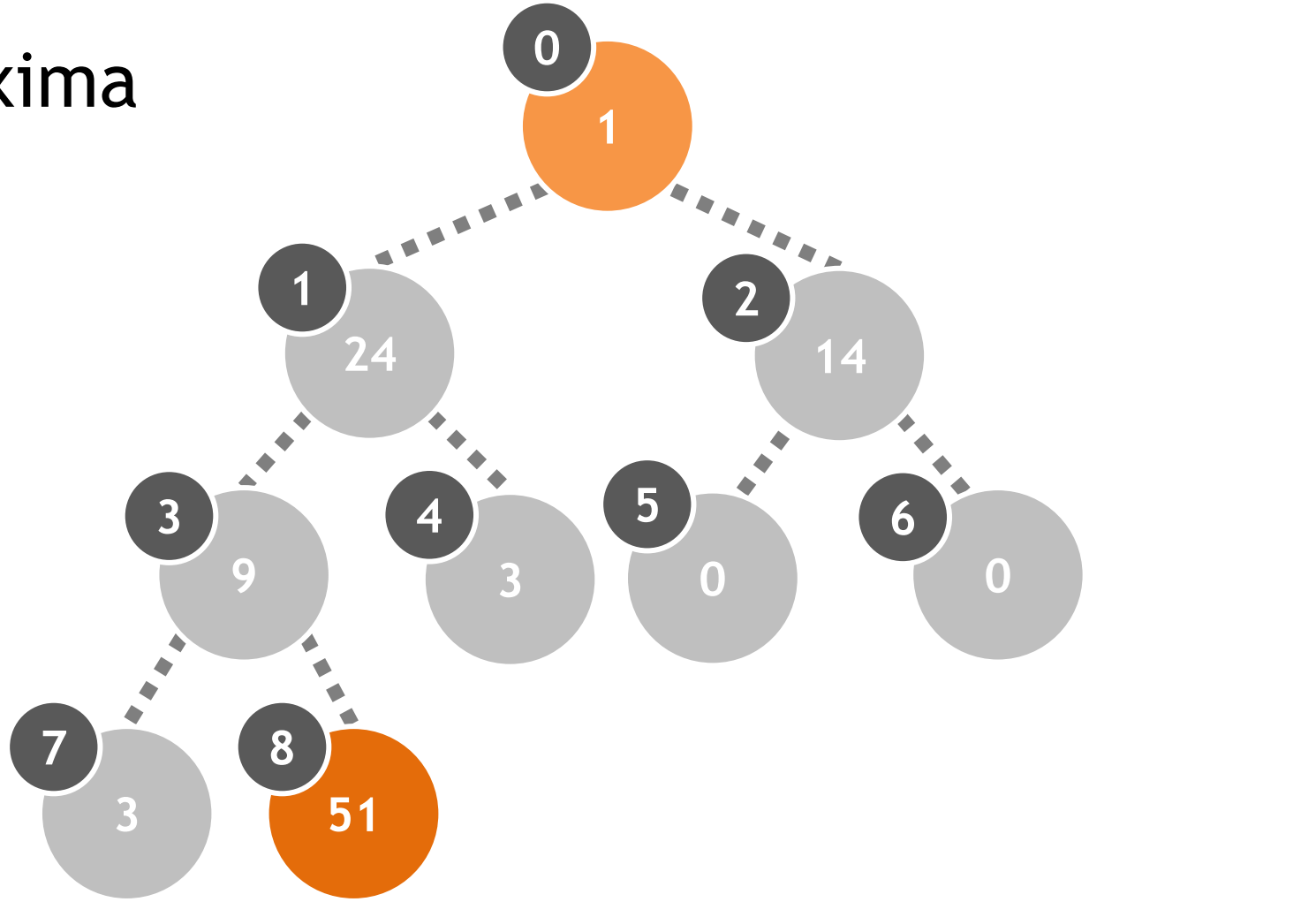


51	24	14	9	3	0	0	3	1
0	1	2	3	4	5	6	7	8

Extrakcia maxima

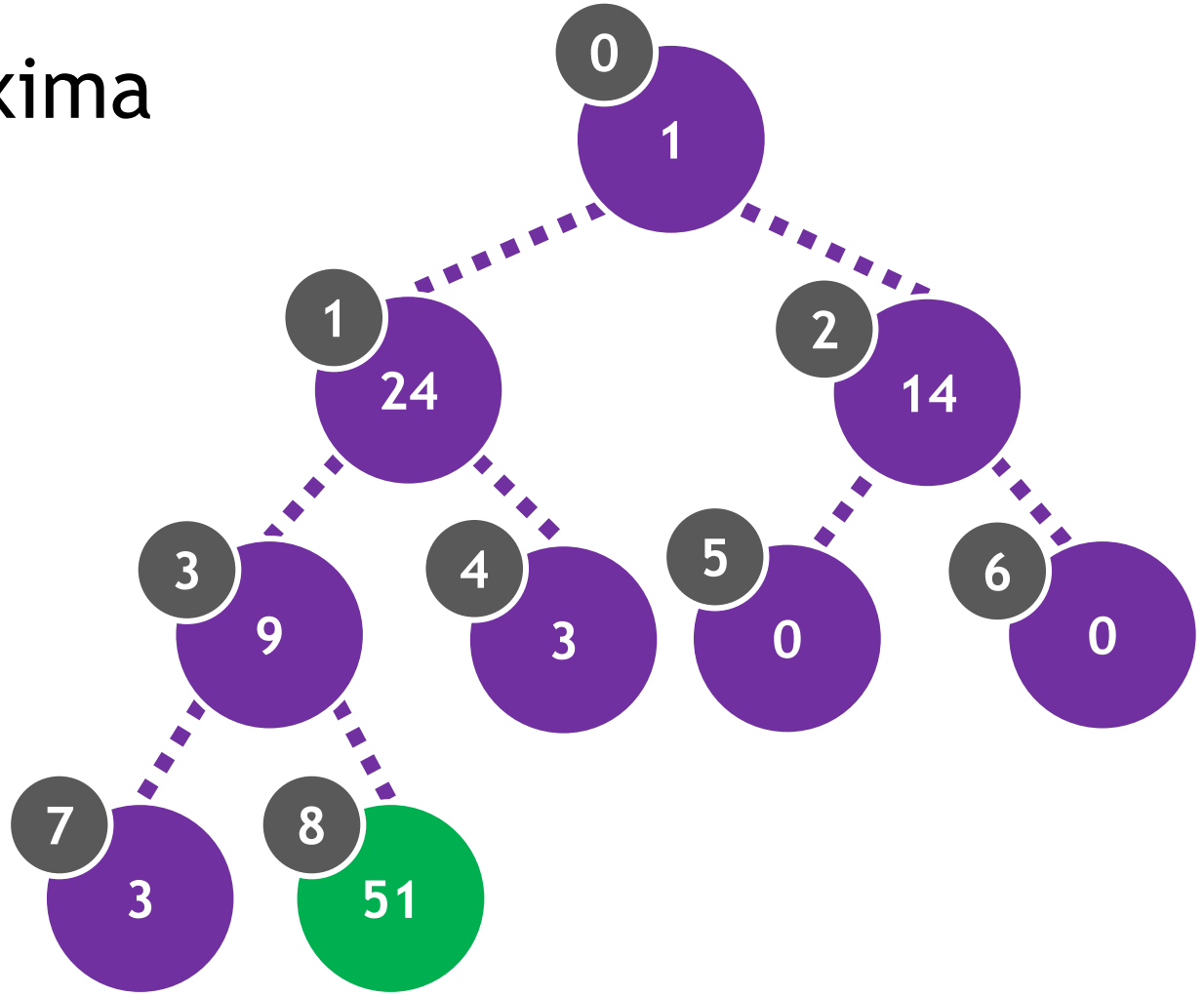


Extrakcia maxima



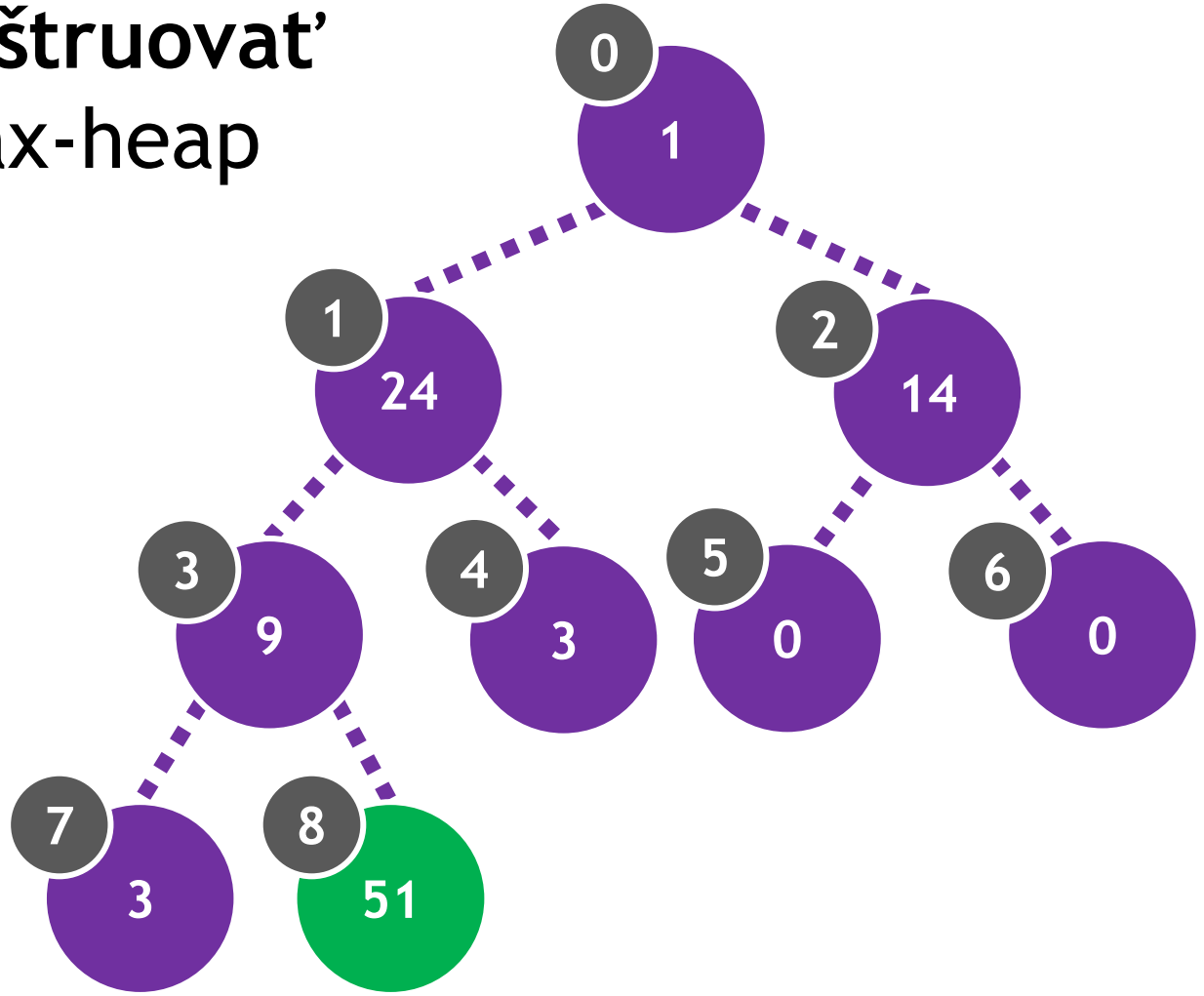
1	24	14	9	3	0	0	3	51
0	1	2	3	4	5	6	7	8

Extrakcia maxima



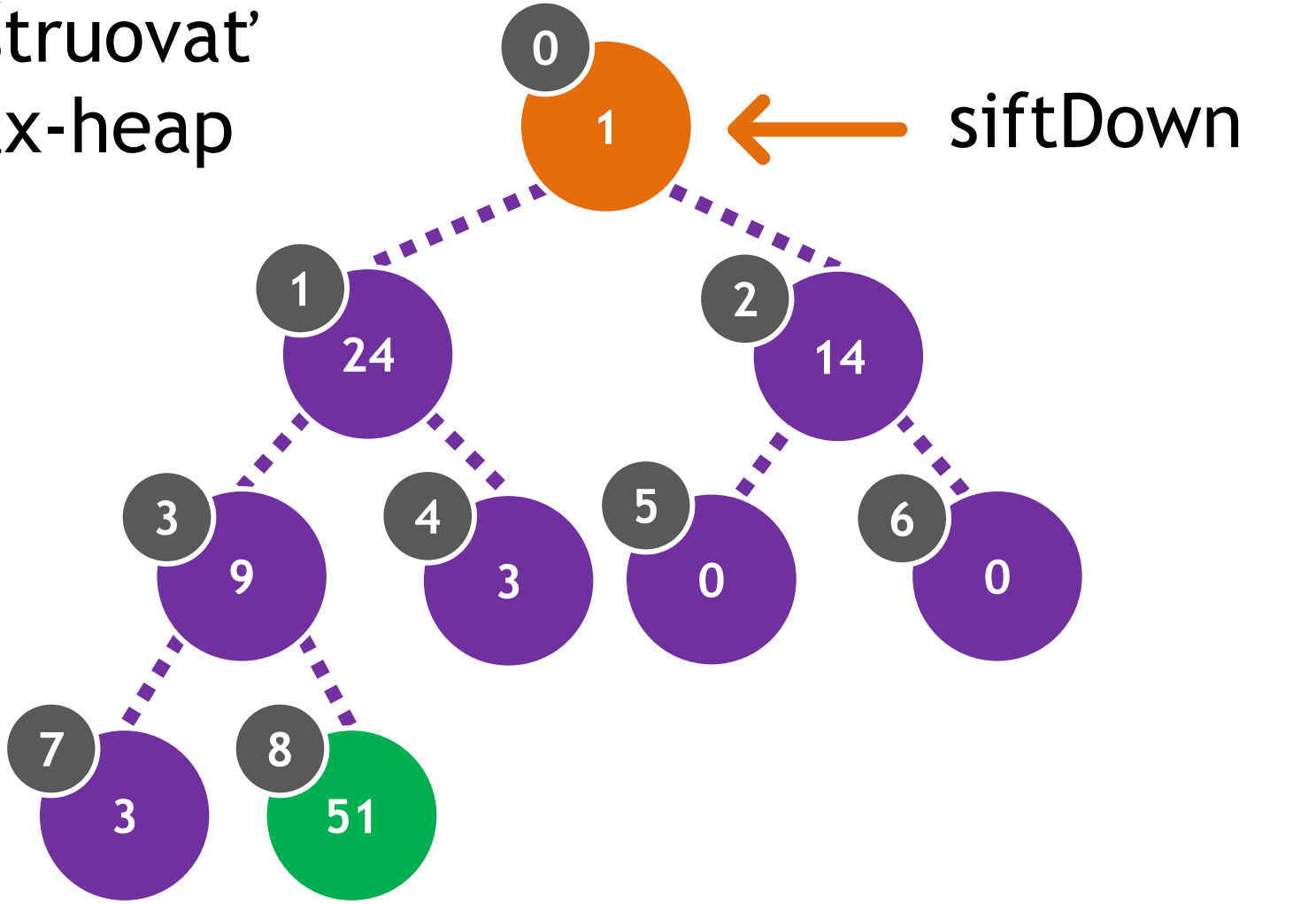
1	24	14	9	3	0	0	3	51
0	1	2	3	4	5	6	7	8

Treba **zrekonštruovať** poškodený Max-heap



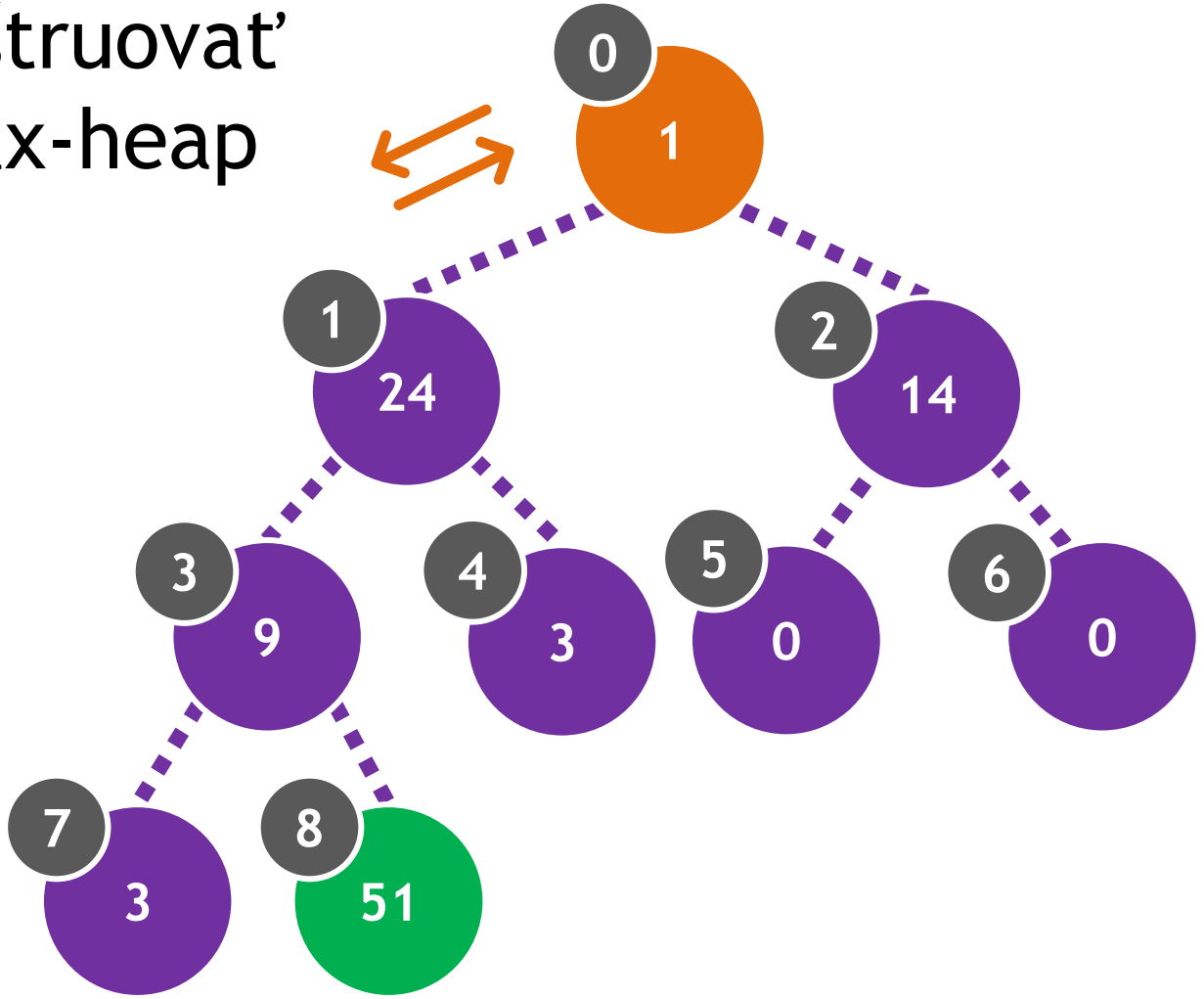
1	24	14	9	3	0	0	3	51
0	1	2	3	4	5	6	7	8

Treba zrekonštruovať poškodený Max-heap



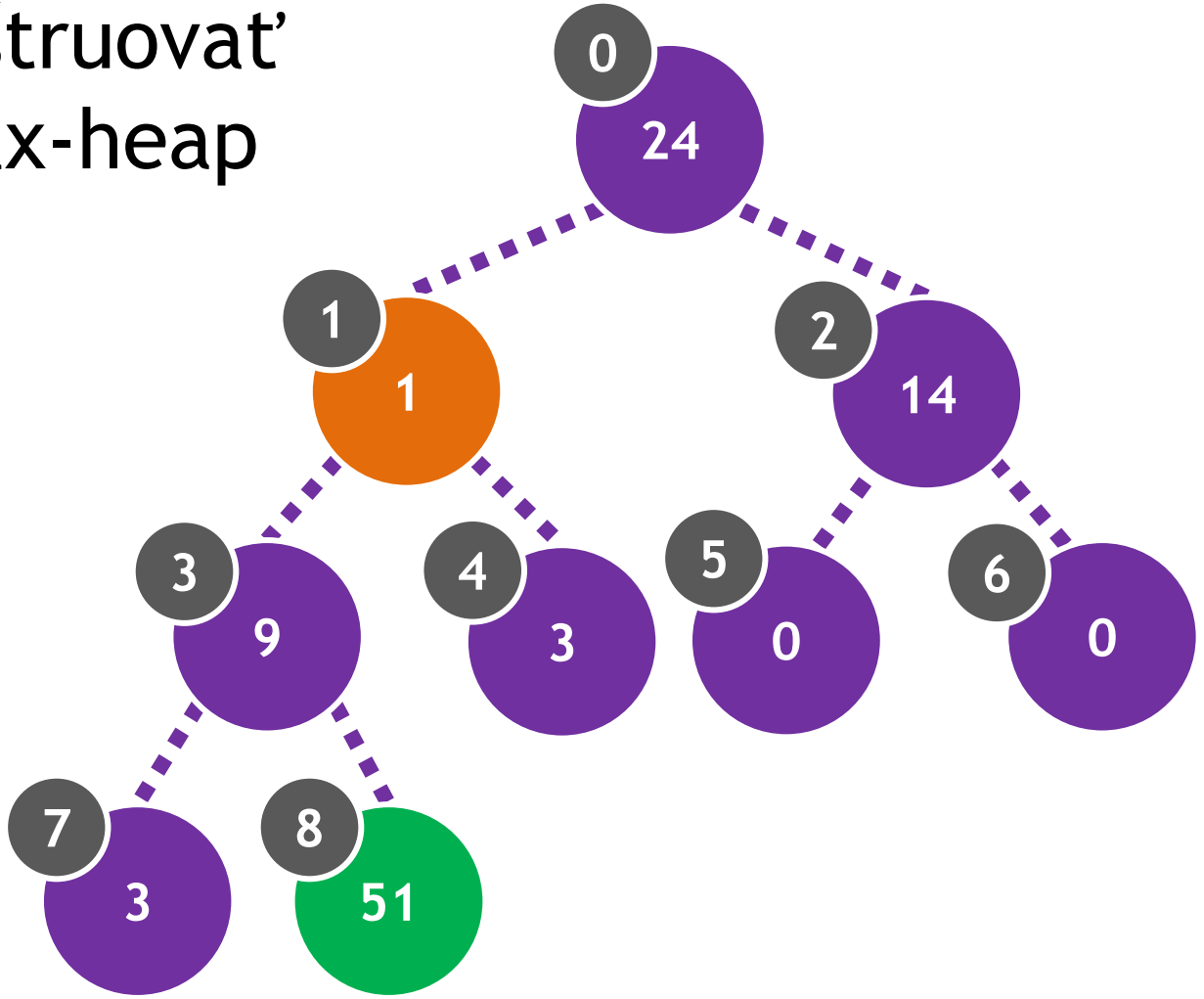
1	24	14	9	3	0	0	3	51
0	1	2	3	4	5	6	7	8

Treba zrekonštruovať poškodený Max-heap



1	24	14	9	3	0	0	3	51
0	1	2	3	4	5	6	7	8

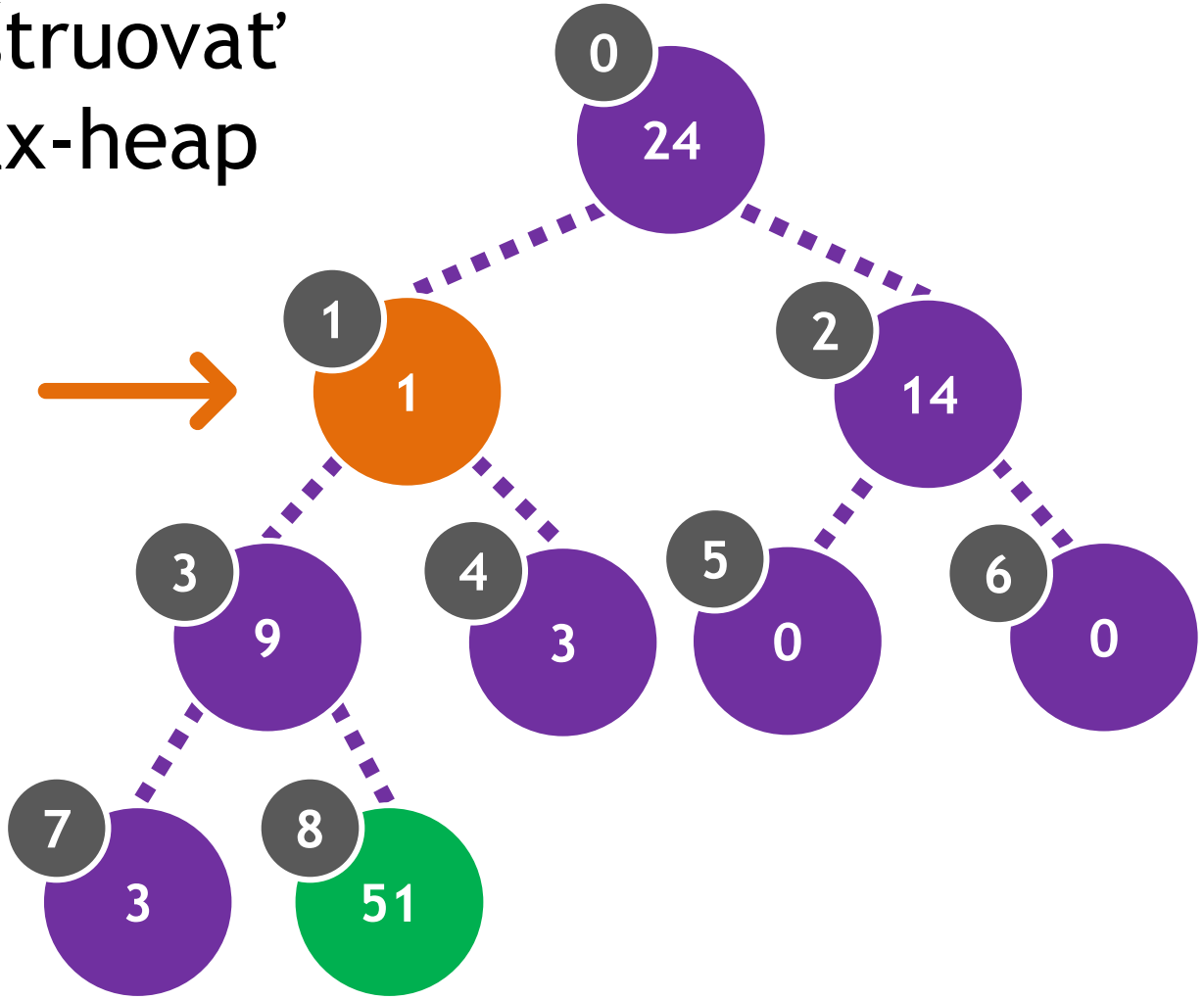
Treba zrekonštruovať poškodený Max-heap



24	1	14	9	3	0	0	3	51
0	1	2	3	4	5	6	7	8

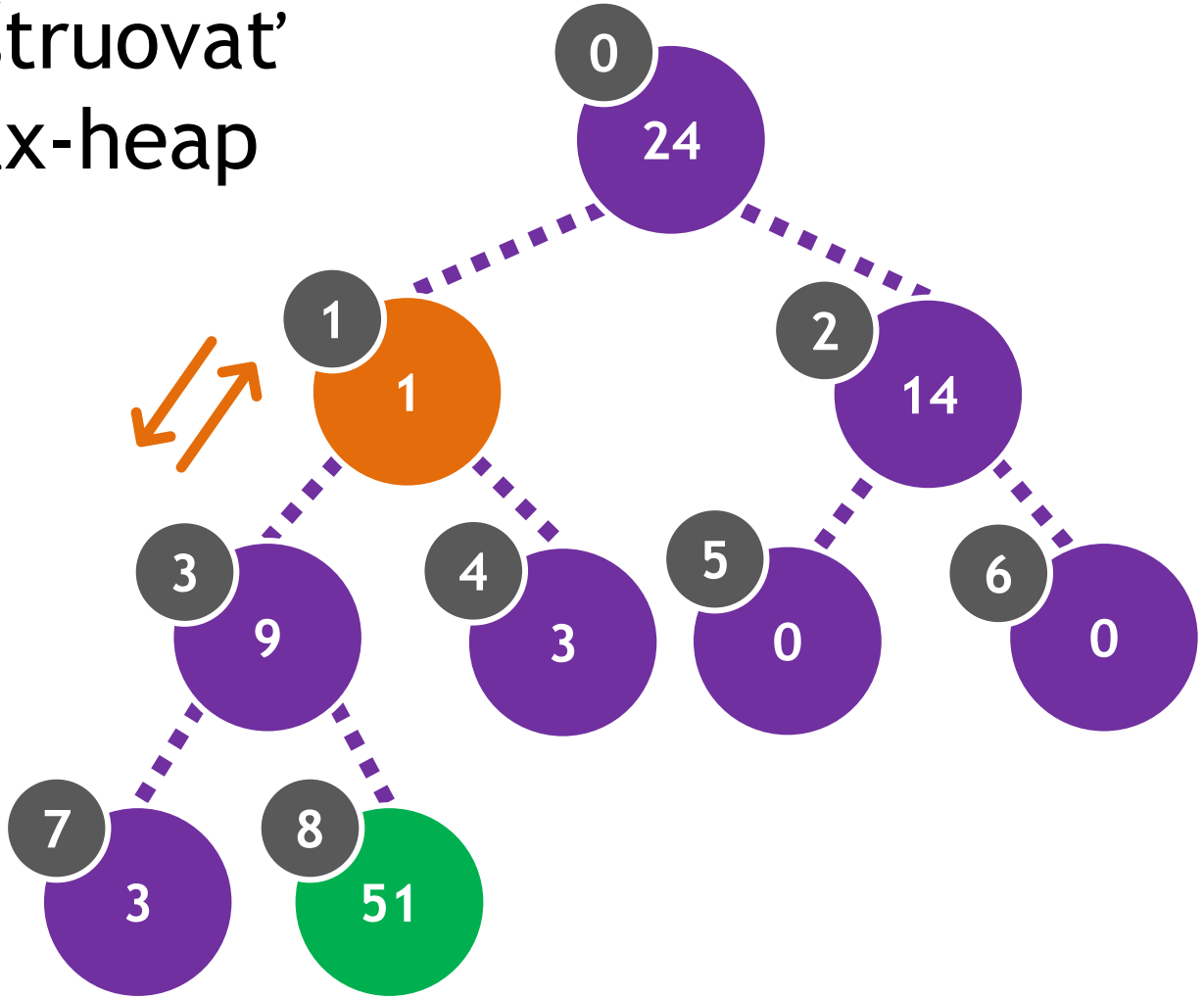
Treba zrekonštruovať poškodený Max-heap

siftDown →



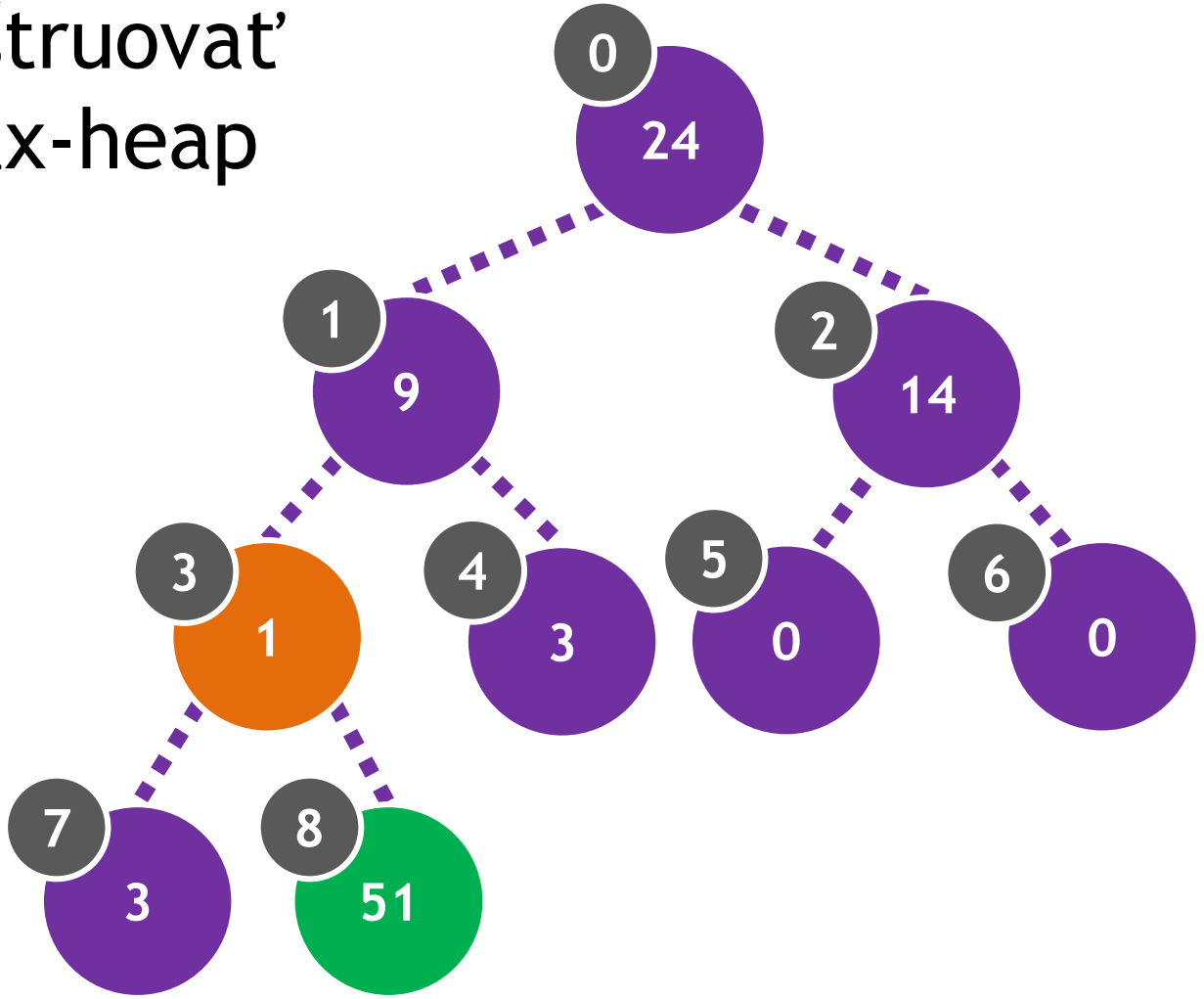
24	1	14	9	3	0	0	3	51
0	1	2	3	4	5	6	7	8

Treba zrekonštruovať poškodený Max-heap



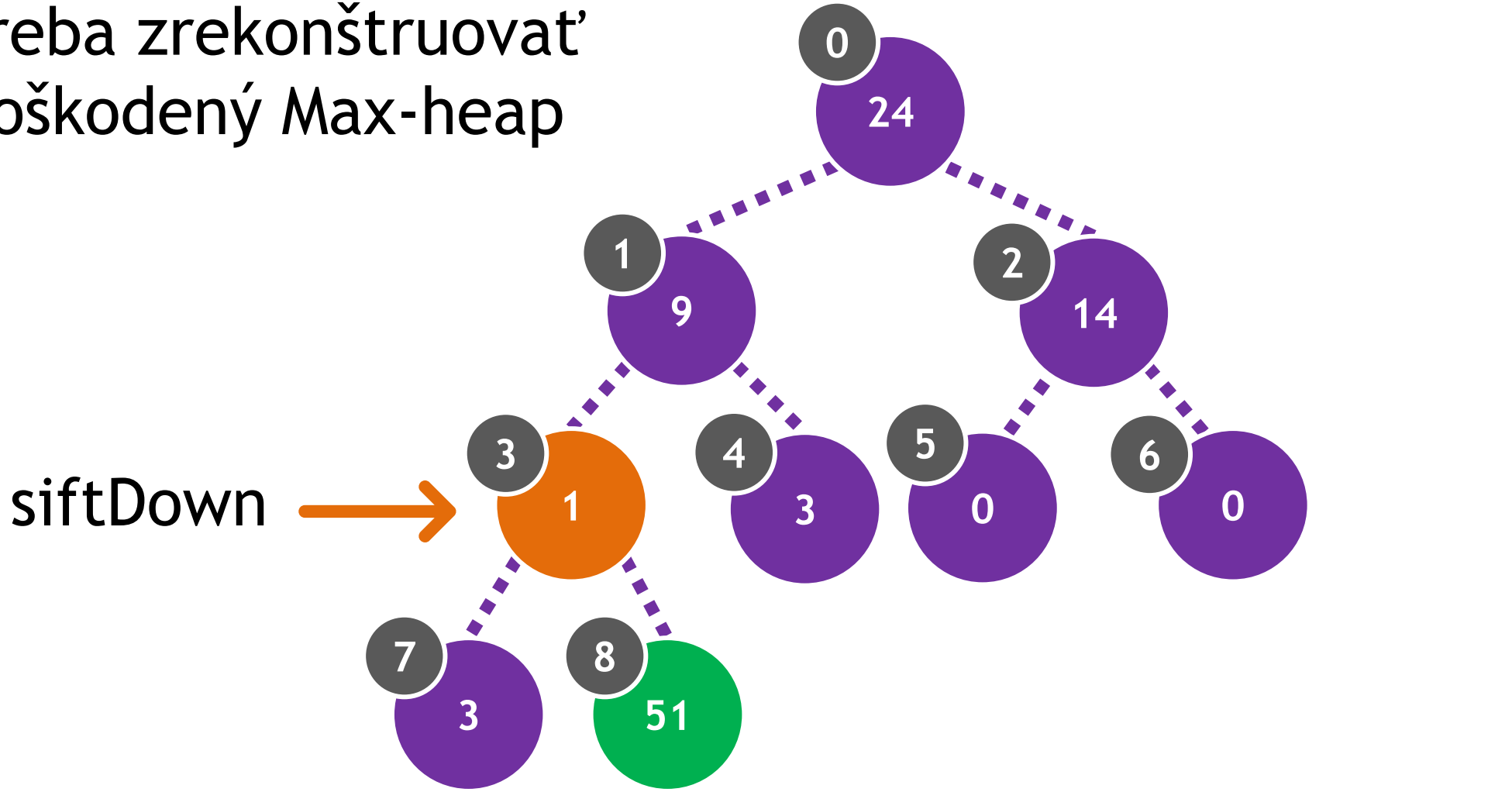
24	1	14	9	3	0	0	3	51
0	1	2	3	4	5	6	7	8

Treba zrekonštruovať poškodený Max-heap



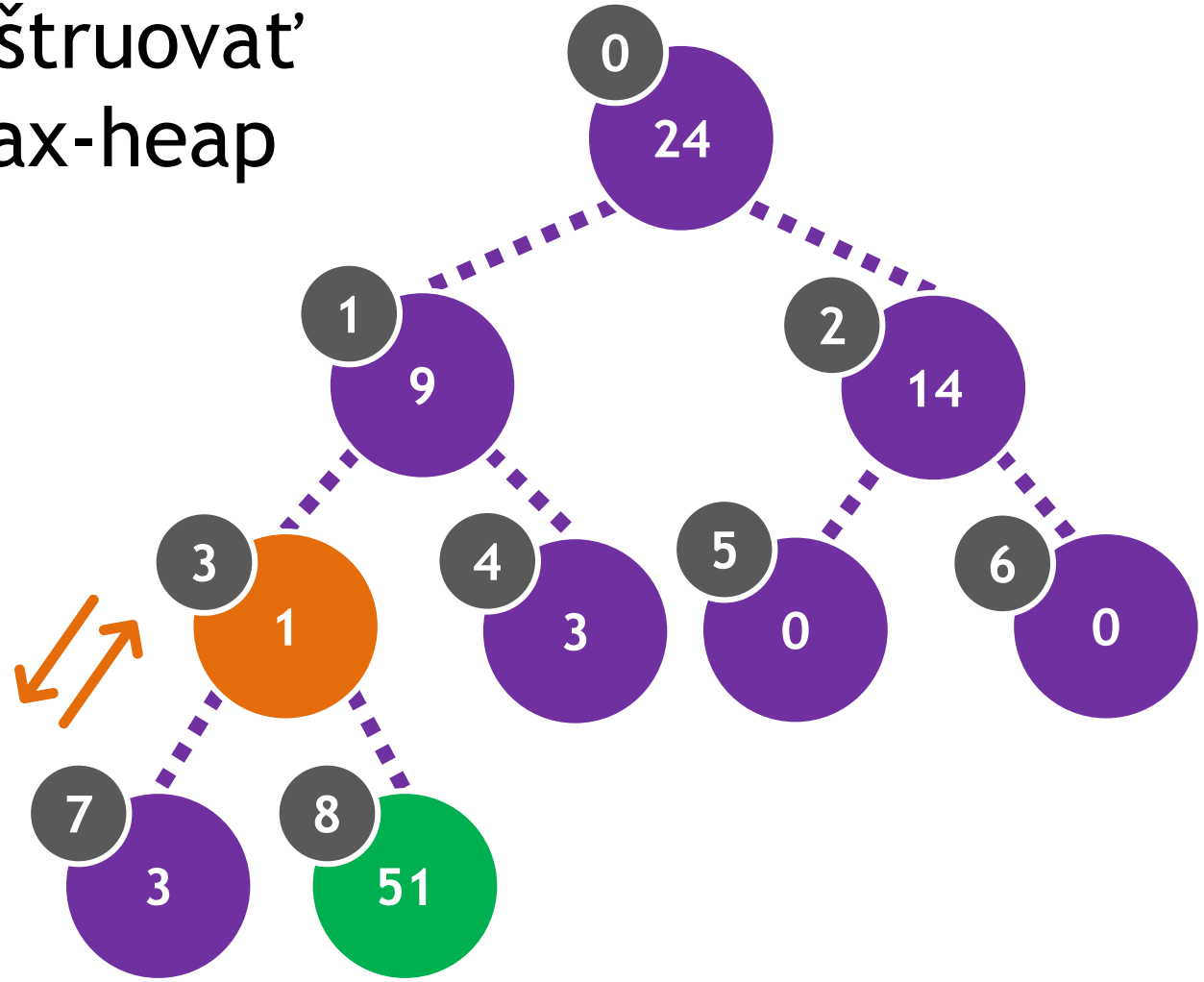
24	9	14	1	3	0	0	3	51
0	1	2	3	4	5	6	7	8

Treba zrekonštruovať poškodený Max-heap



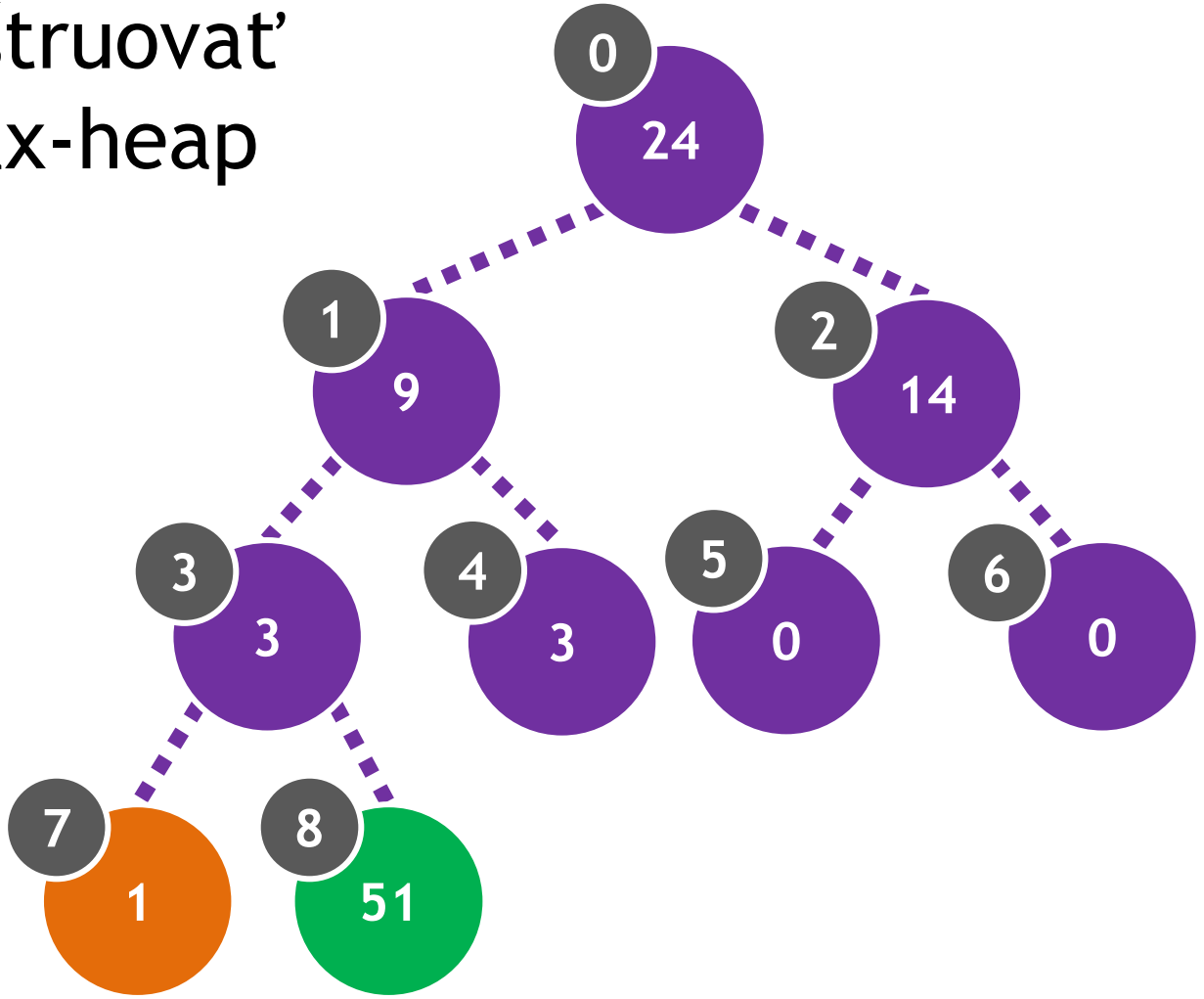
24	9	14	1	3	0	0	3	51
0	1	2	3	4	5	6	7	8

Treba zrekonštruovať poškodený Max-heap



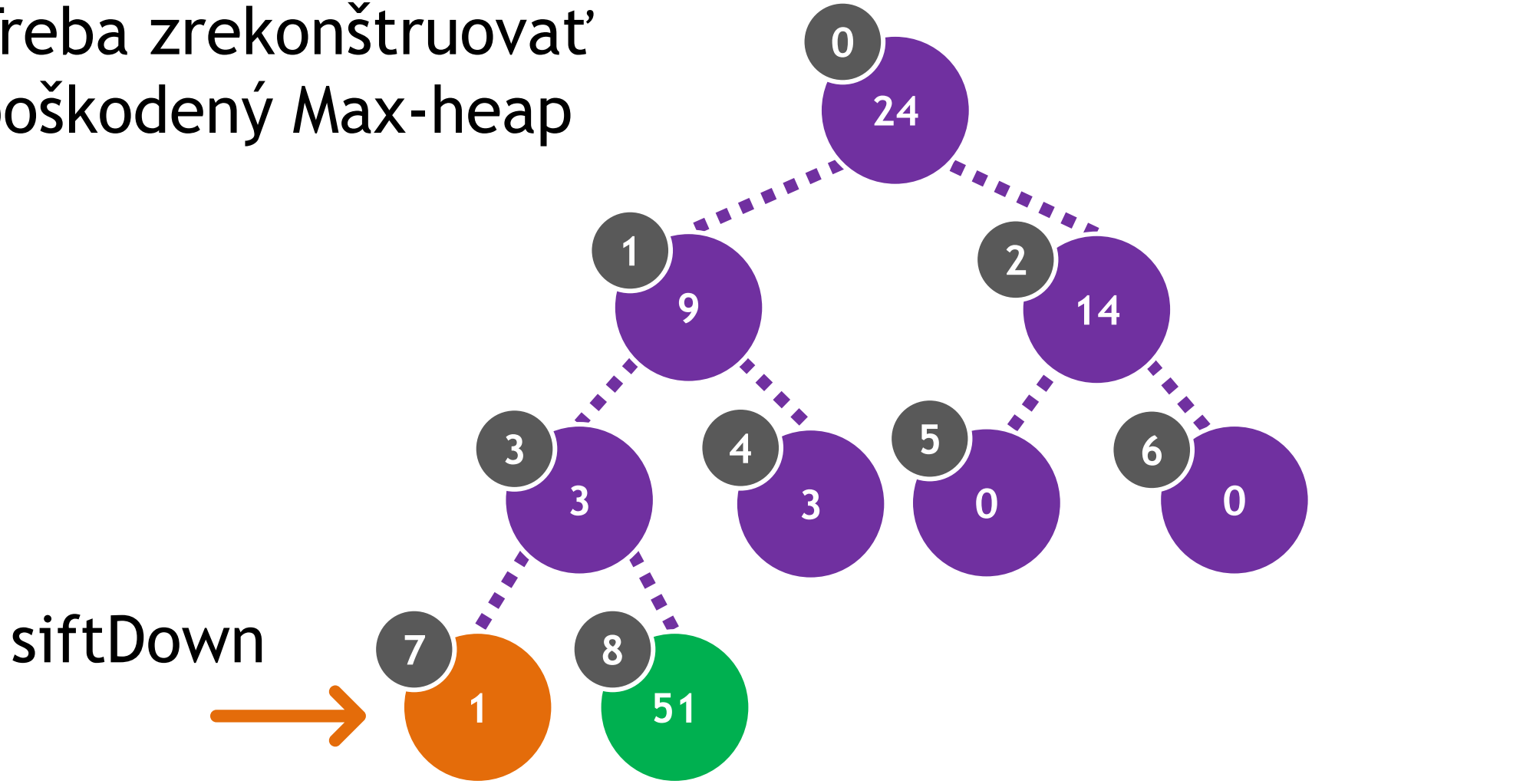
24	9	14	1	3	0	0	3	51
0	1	2	3	4	5	6	7	8

Treba zrekonštruovať poškodený Max-heap



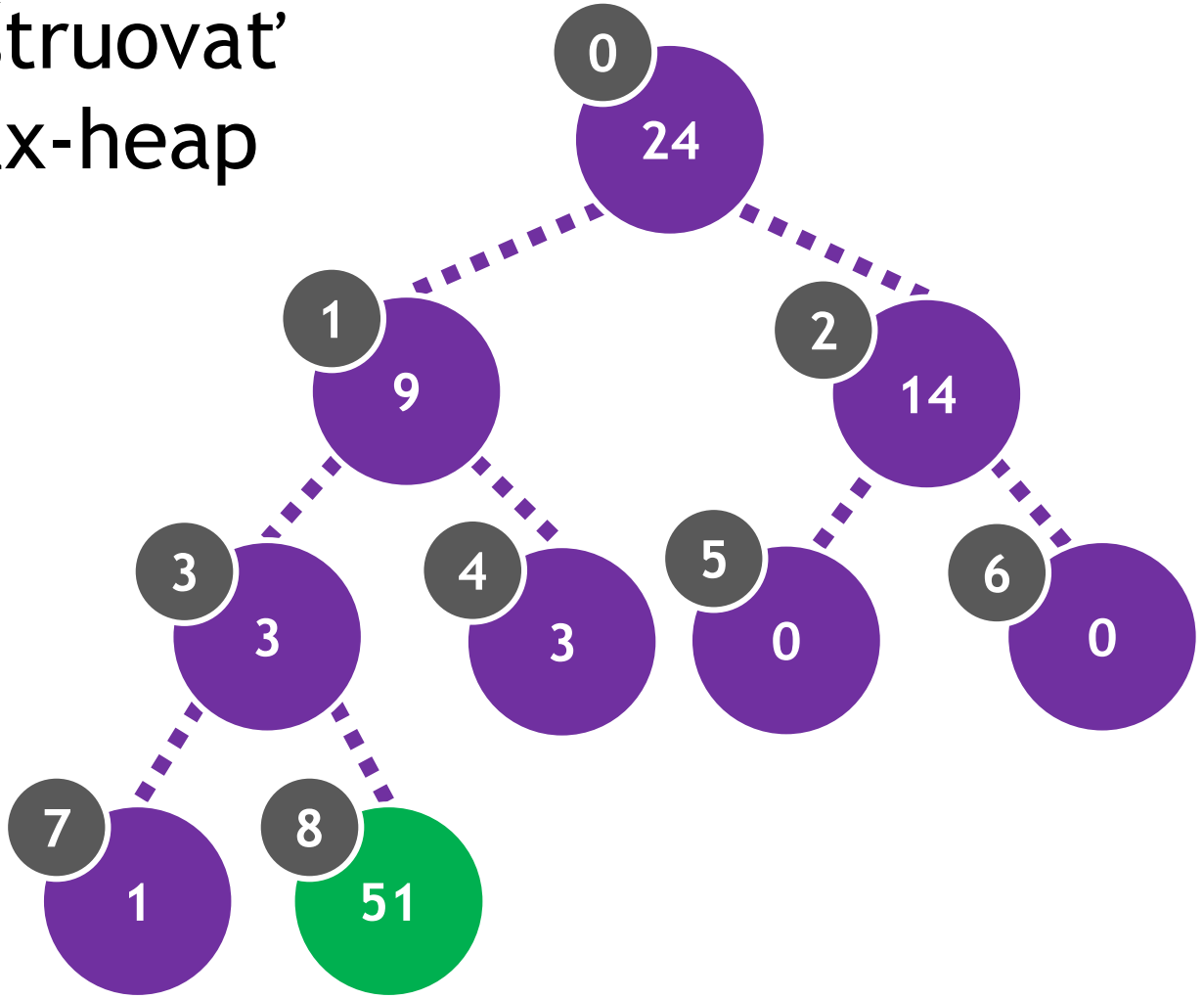
24	9	14	3	3	0	0	1	51
0	1	2	3	4	5	6	7	8

Treba zrekonštruovať poškodený Max-heap



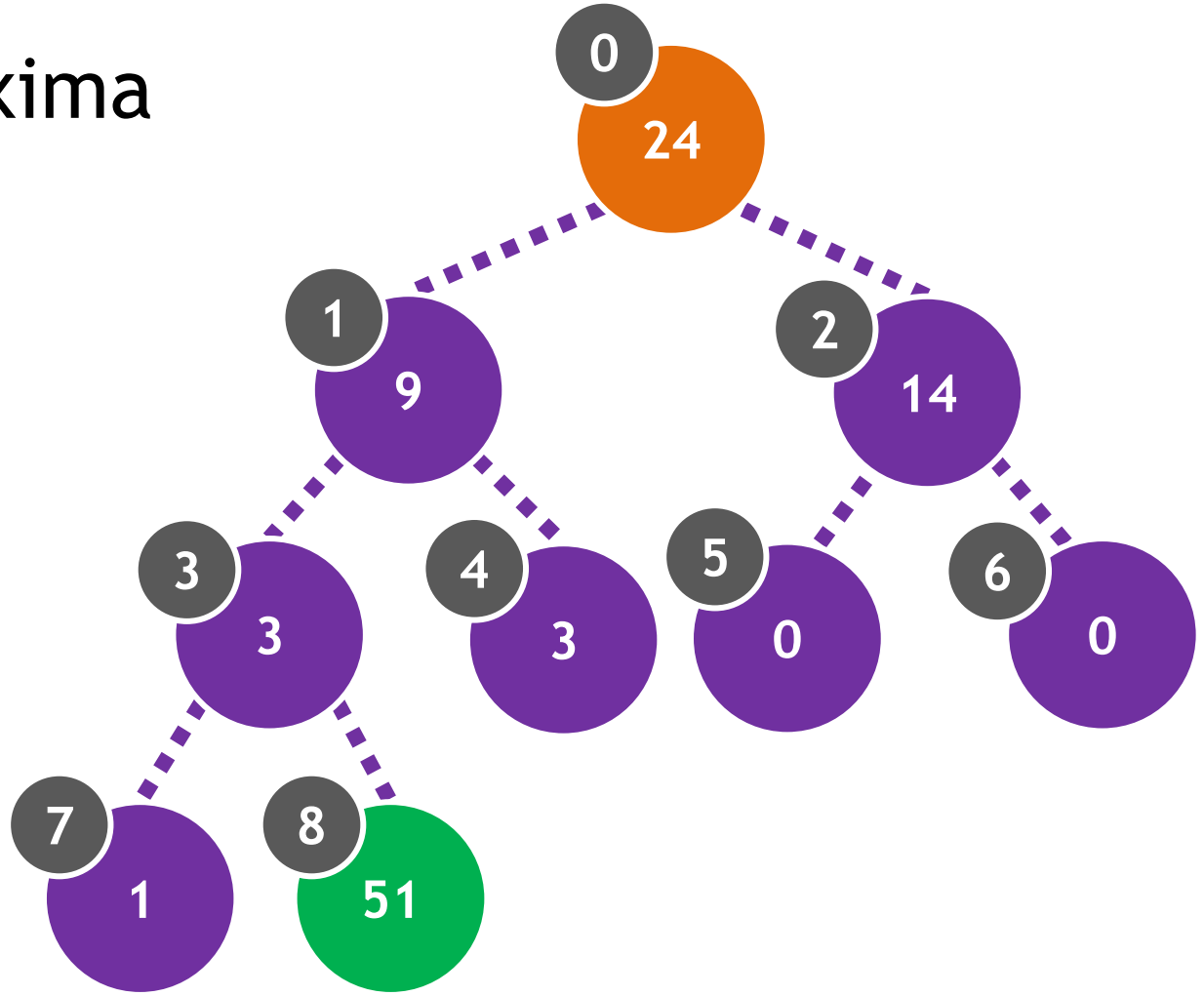
24	9	14	3	3	0	0	1	51
0	1	2	3	4	5	6	7	8

Treba zrekonštruovať poškodený Max-heap

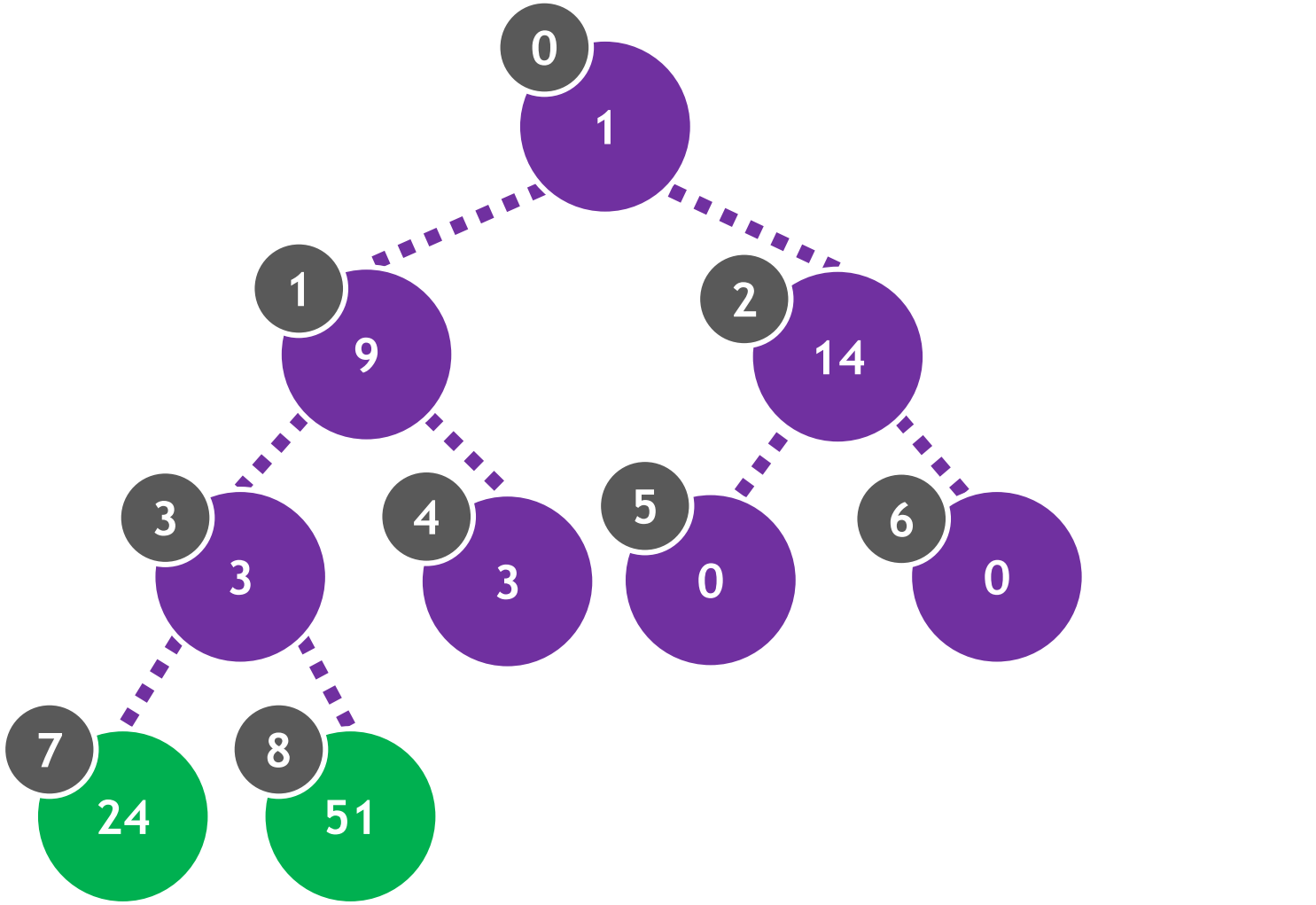


24	9	14	3	3	0	0	1	51
0	1	2	3	4	5	6	7	8

Extrakcia maxima

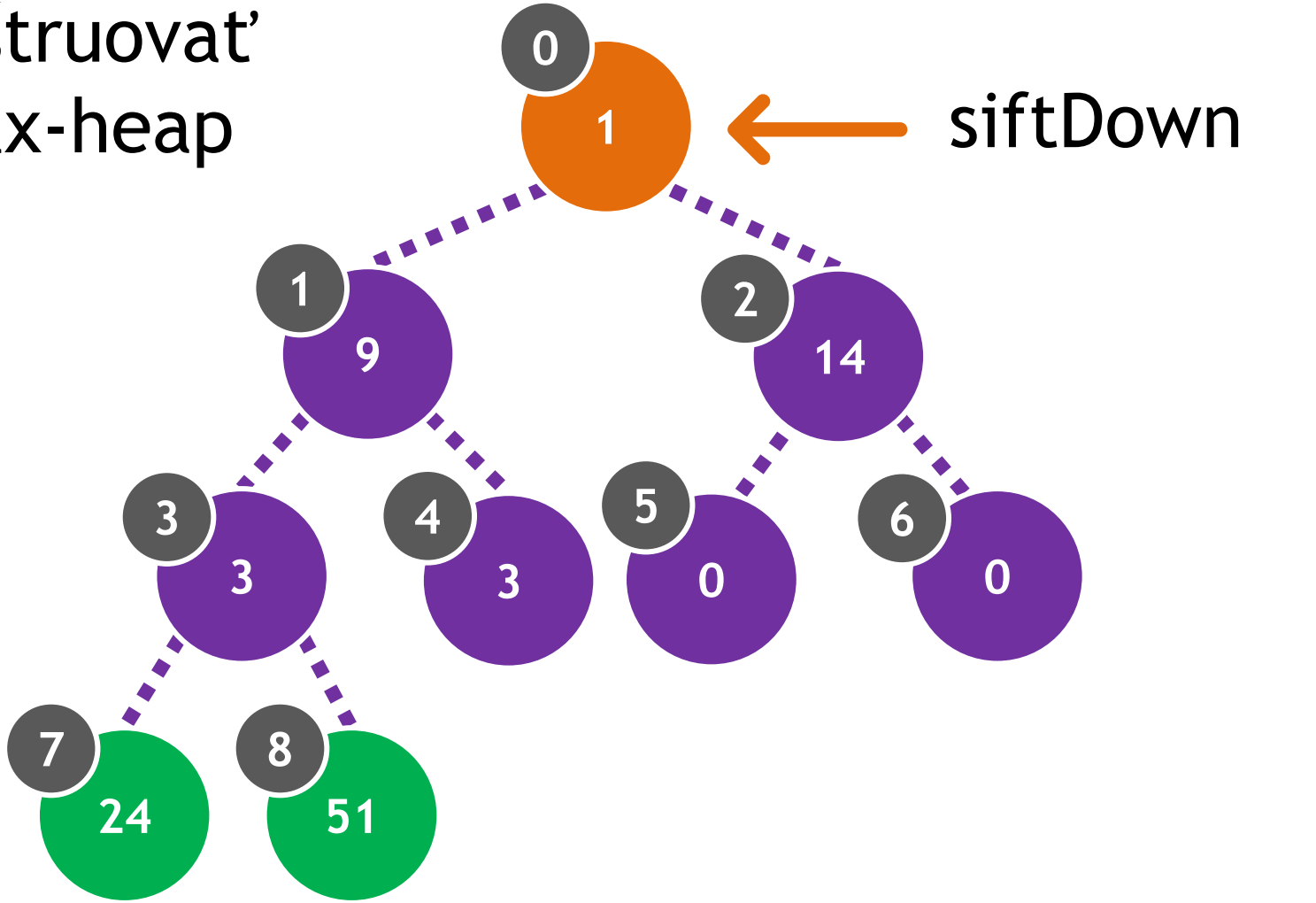


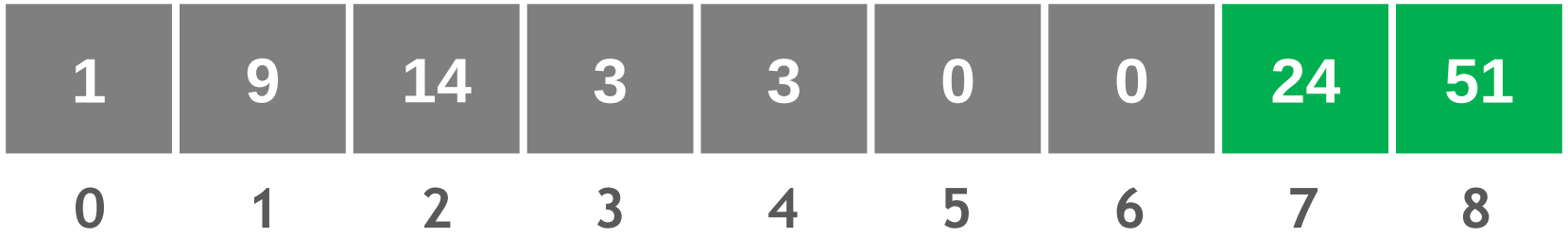
24	9	14	3	3	0	0	1	51
0	1	2	3	4	5	6	7	8



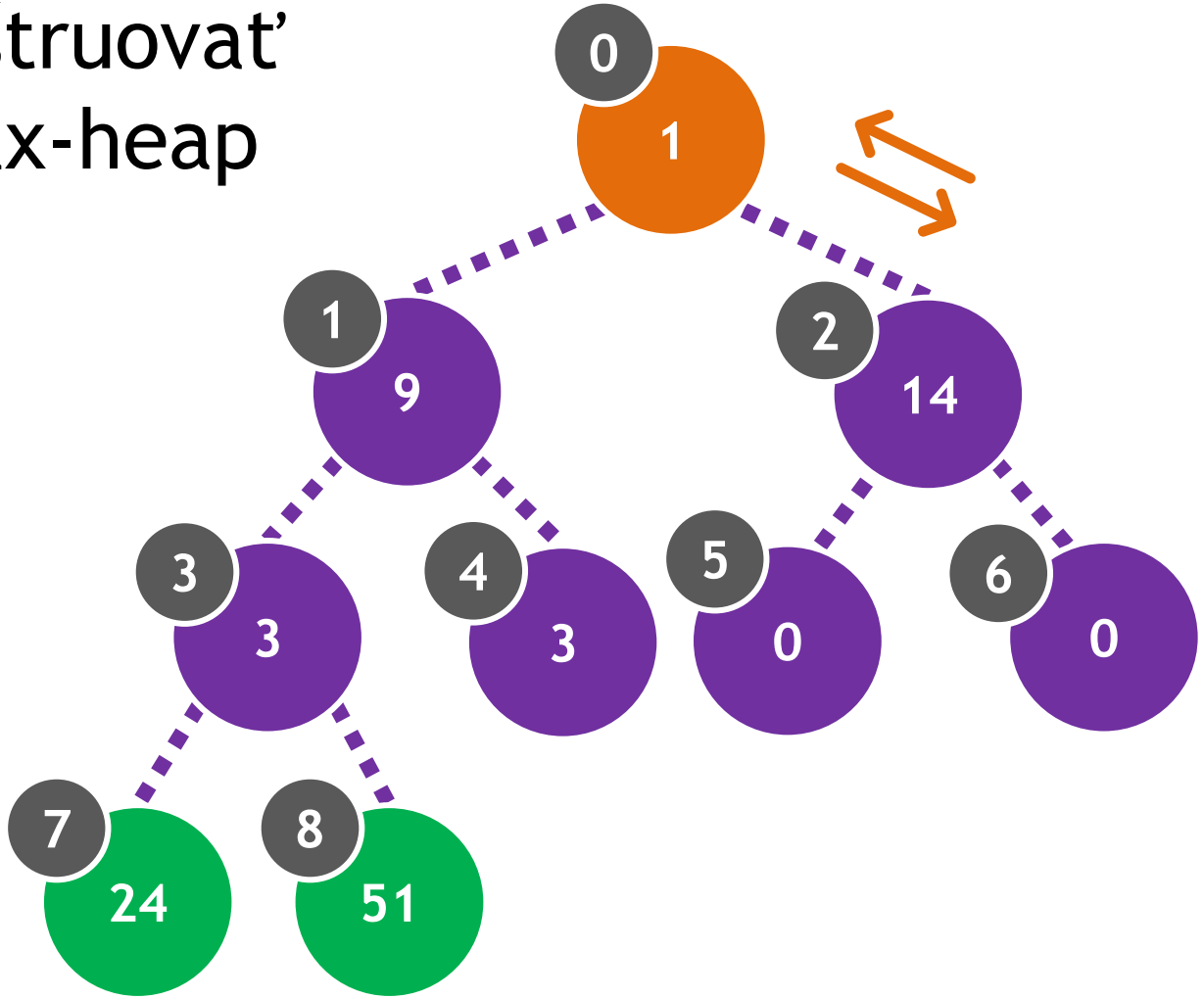
1	9	14	3	3	0	0	24	51
0	1	2	3	4	5	6	7	8

Treba zrekonštruovať poškodený Max-heap



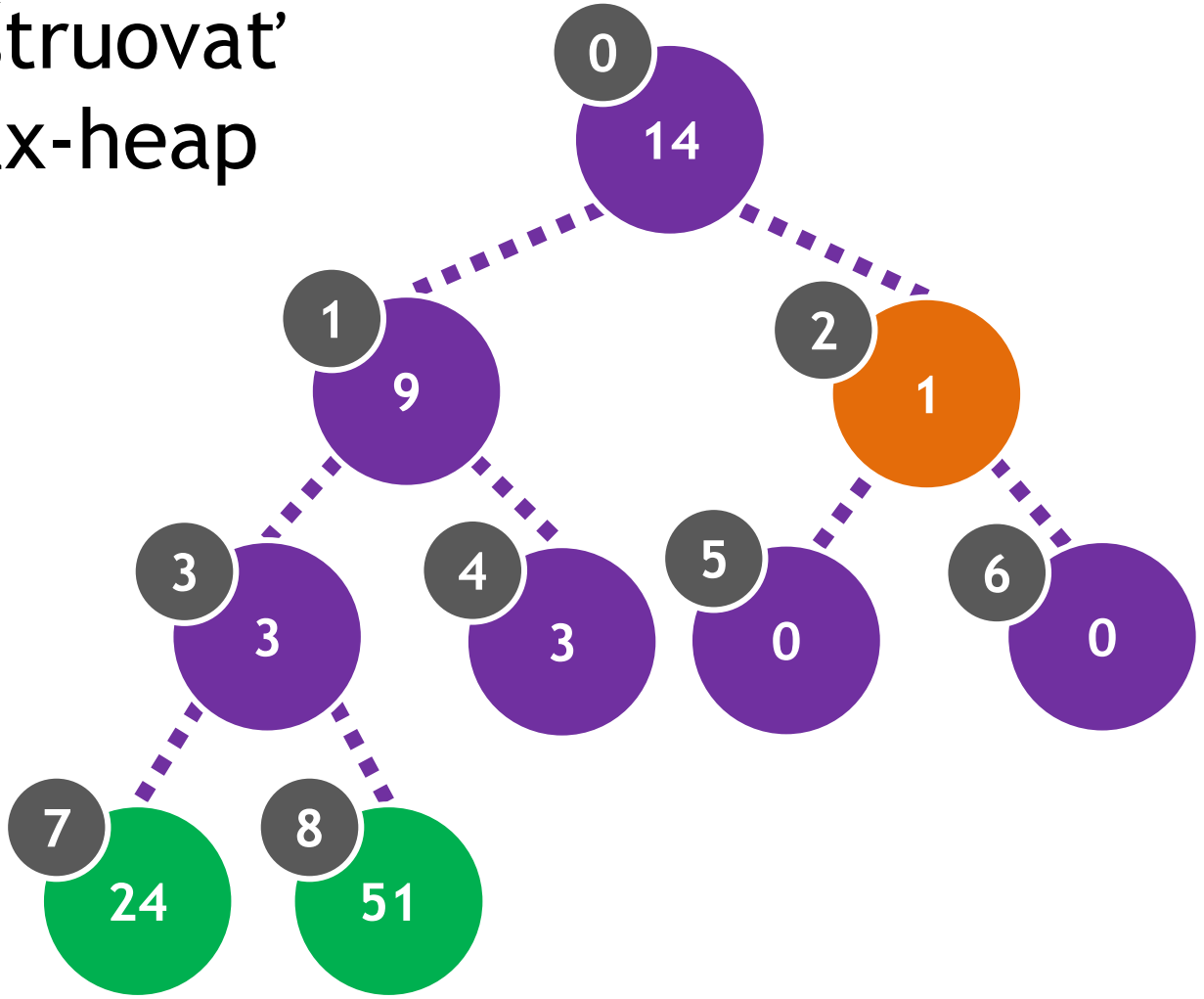


Treba zrekonštruovať poškodený Max-heap



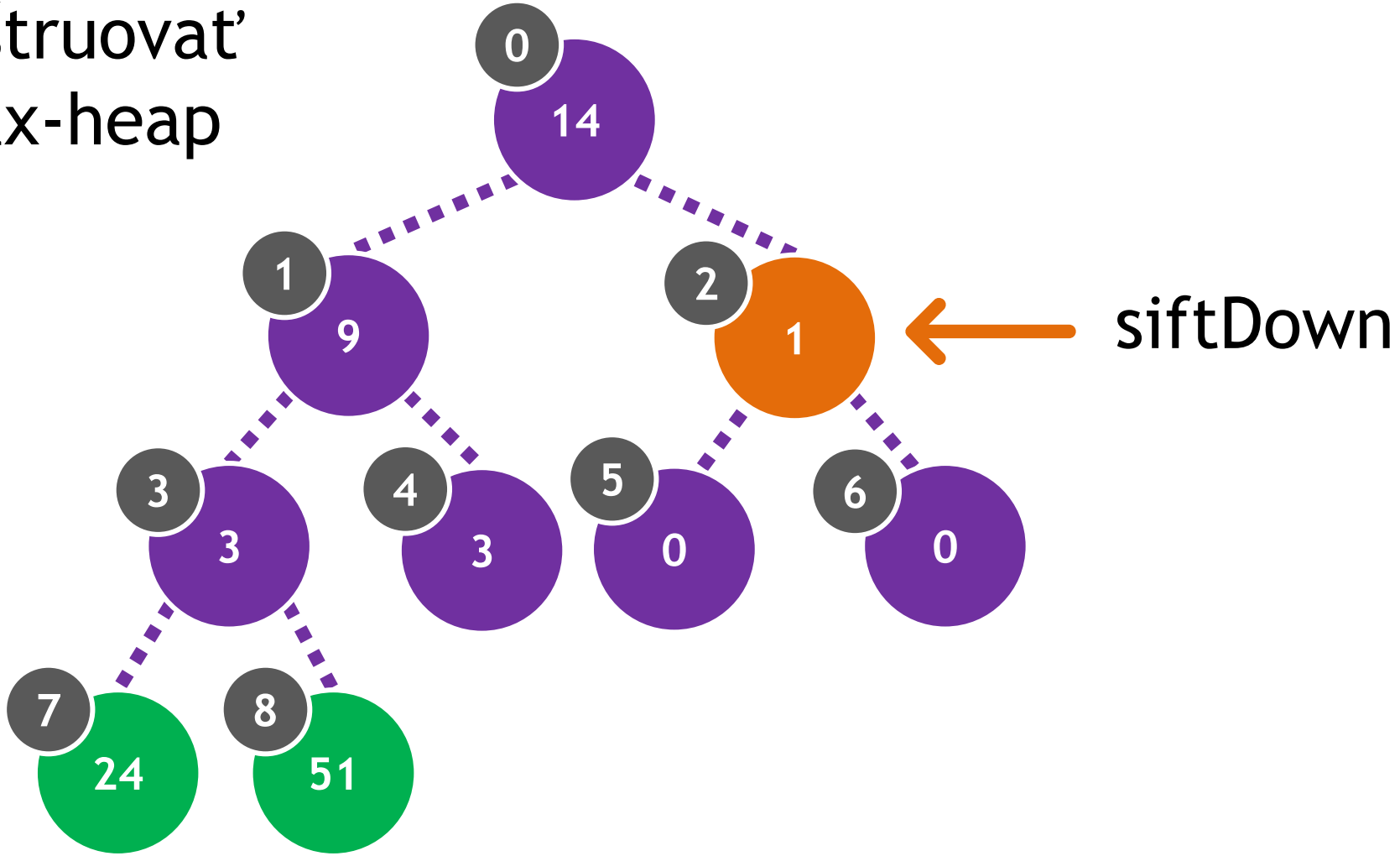
1	9	14	3	3	0	0	24	51
0	1	2	3	4	5	6	7	8

Treba zrekonštruovať poškodený Max-heap



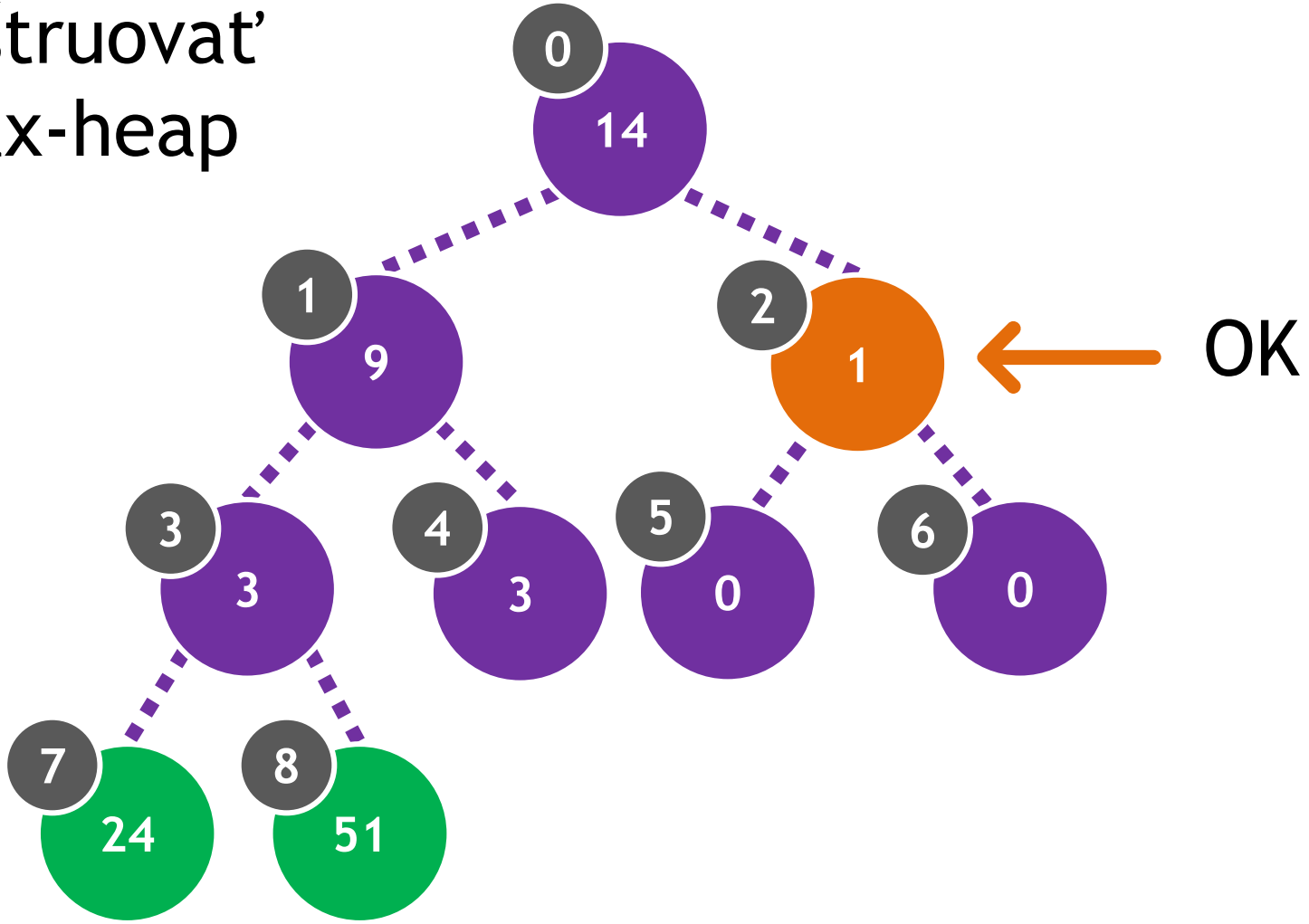
14	9	1	3	3	0	0	24	51
0	1	2	3	4	5	6	7	8

Treba zrekonštruovať poškodený Max-heap



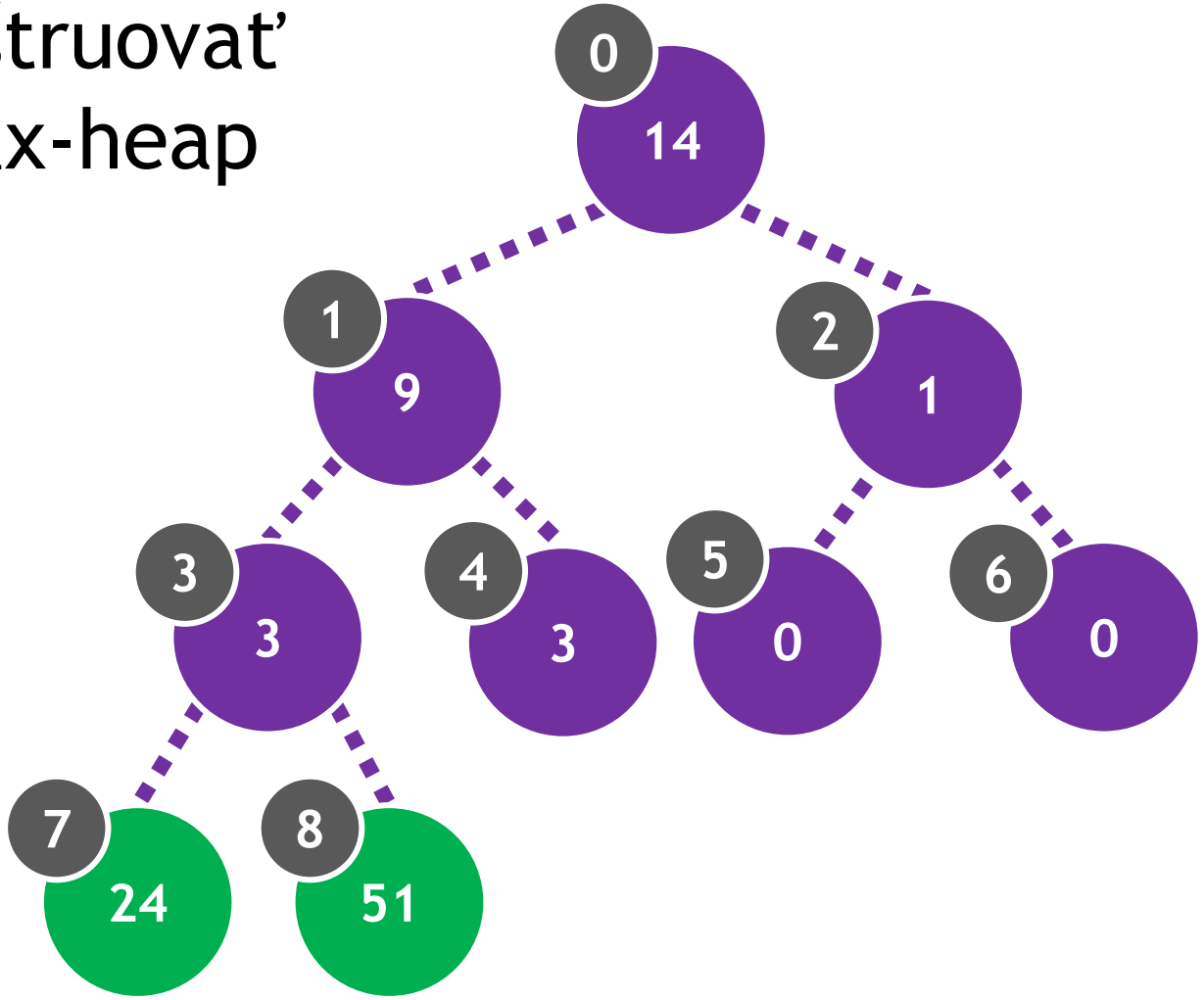
14	9	1	3	3	0	0	24	51
0	1	2	3	4	5	6	7	8

Treba zrekonštruovať poškodený Max-heap



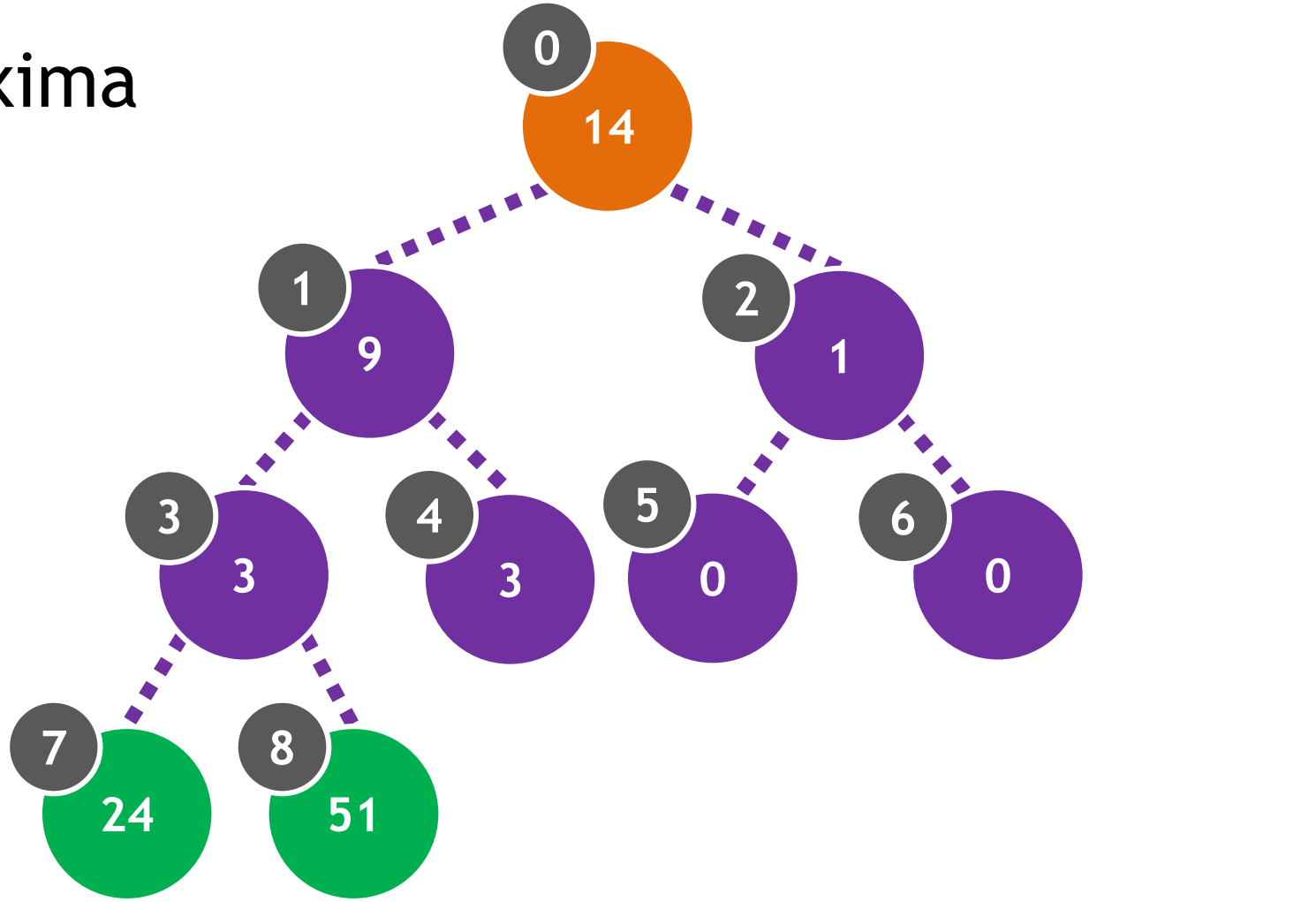
14	9	1	3	3	0	0	24	51
0	1	2	3	4	5	6	7	8

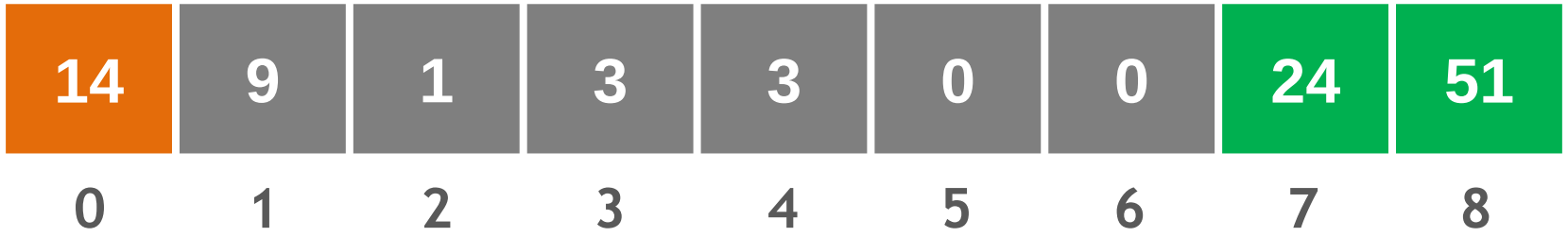
Treba zrekonštruovať poškodený Max-heap



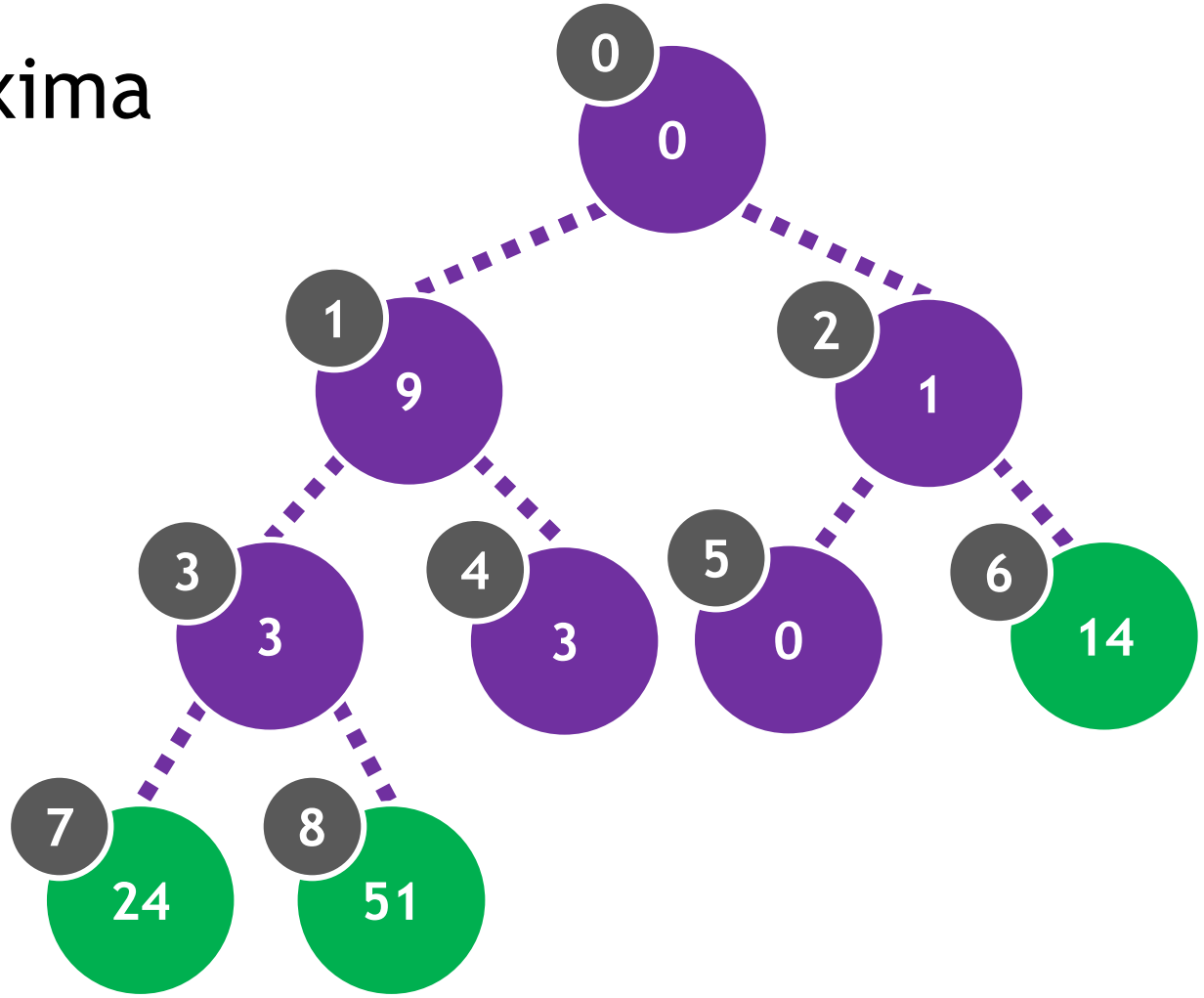
14	9	1	3	3	0	0	24	51
0	1	2	3	4	5	6	7	8

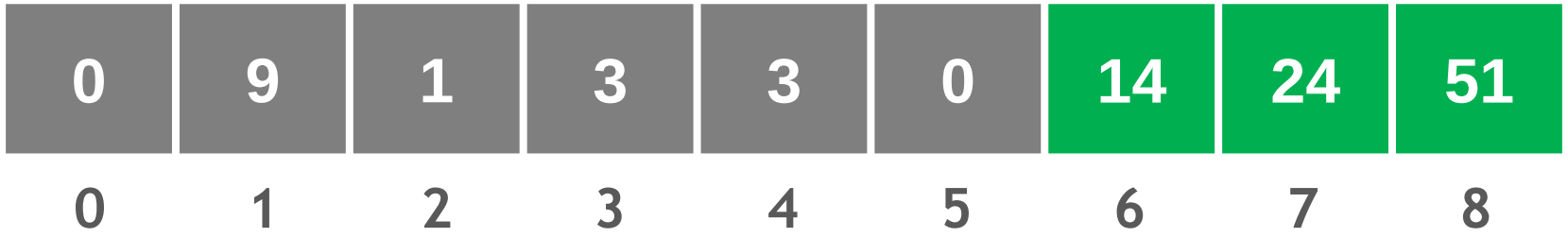
Extrakcia maxima



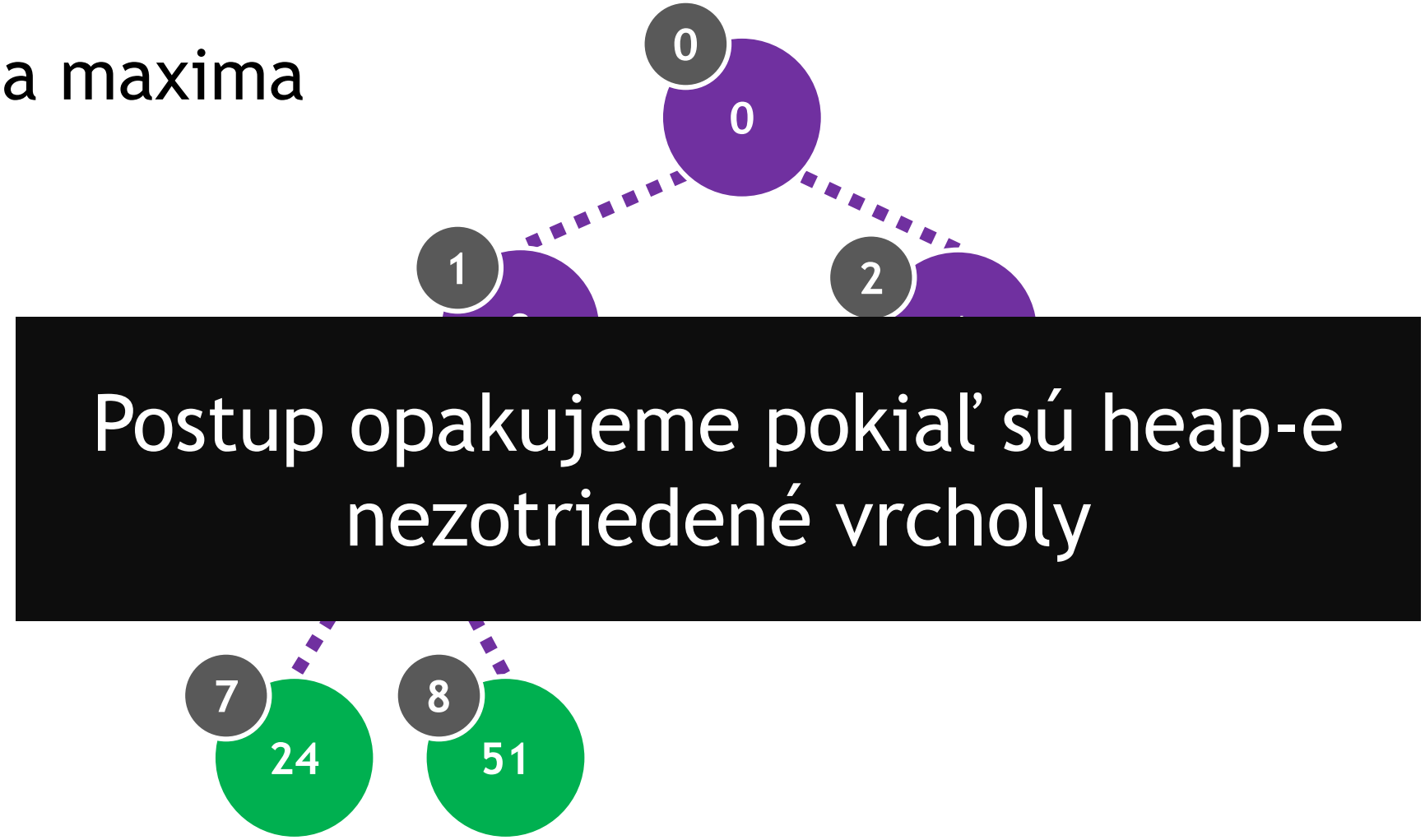


Extrakcia maxima



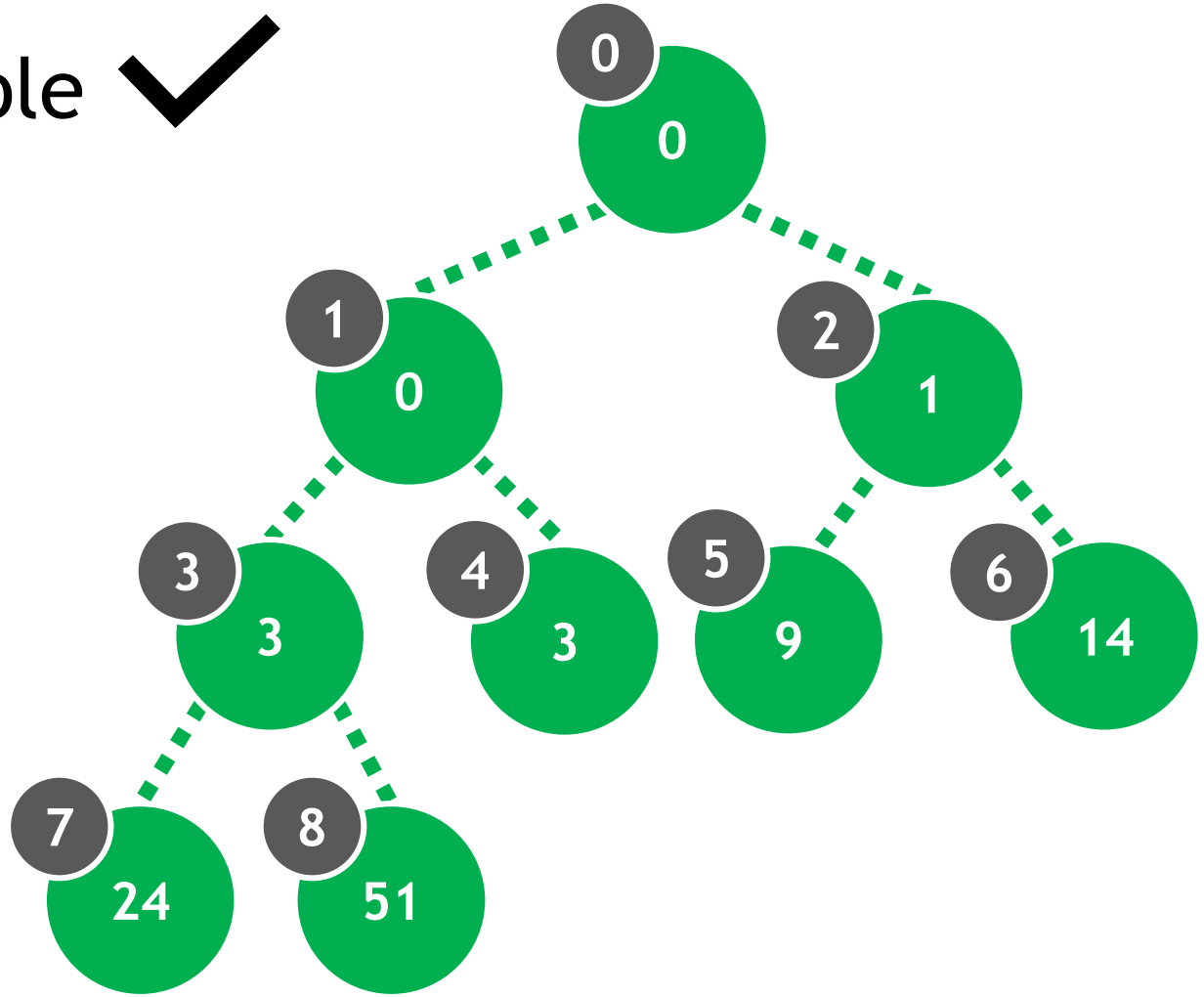


Extrakcia maxima



0	9	1	3	3	0	14	24	51
0	1	2	3	4	5	6	7	8

Zotriedené pole ✓



0	0	1	3	3	9	14	24	51
0	1	2	3	4	5	6	7	8

Vzorová implementácia C/C++