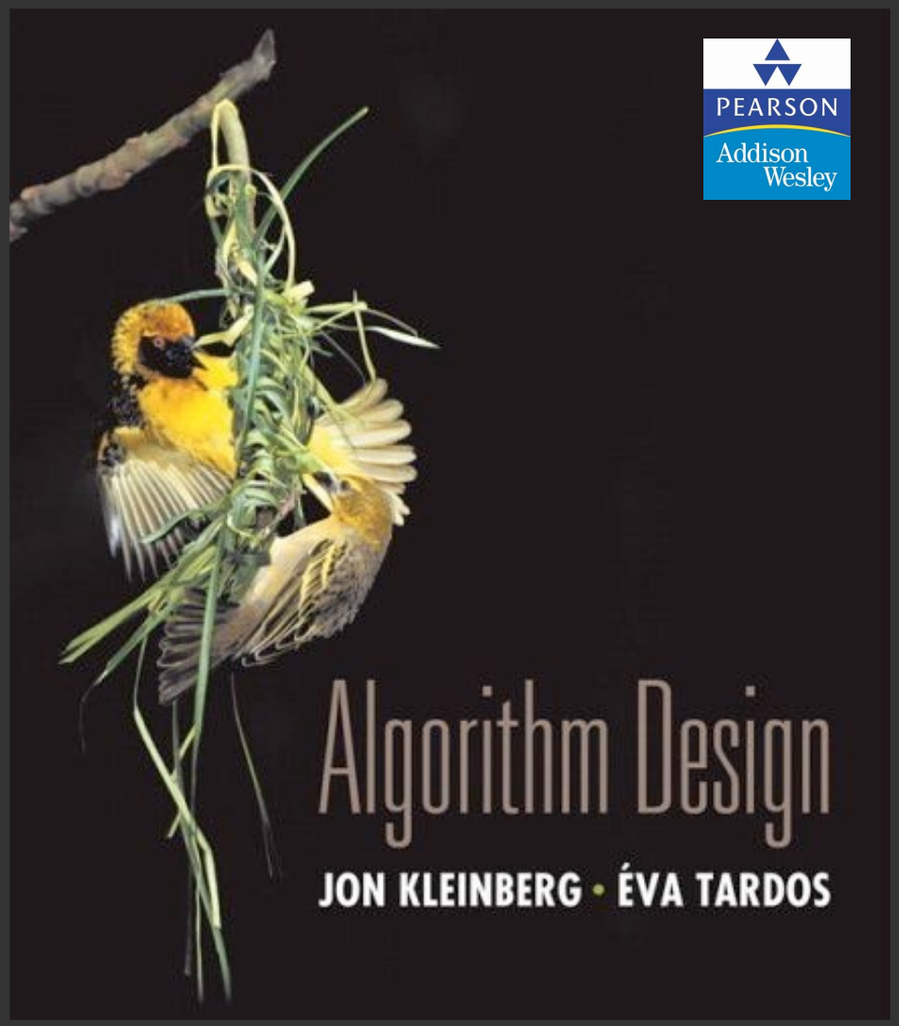


PRIORITY QUEUES

- *binary heaps*

http://www.cs.princeton.edu/~wayne/kleinberg-tardos

Priority queue data type

A min-oriented priority queue supports the following core operations:

- MAKE-HEAP(): create an empty heap.
- INSERT(H, x): insert an element x into the heap.
- EXTRACT-MIN(H): remove and return an element with the smallest key.
- DECREASE-KEY(H, x, k): decrease the key of element x to k.

The following operations are also useful:

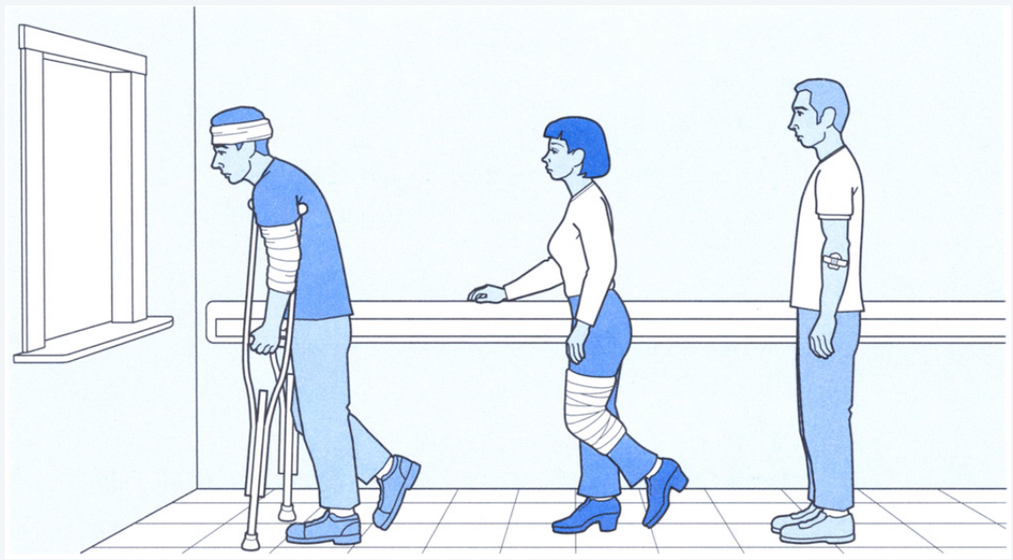
- IS-EMPTY(H): is the heap empty?
- FIND-MIN(H): return an element with smallest key.
- DELETE(H, x): delete element x from the heap.
- MELD(H_1, H_2): replace heaps H_1 and H_2 with their union.

Note. Each element contains a key (duplicate keys are permitted) from a totally-ordered universe.

Priority queue applications

Applications.

- A* search.
- Heapsort.
- Online median.
- Huffman encoding.
- Prim's MST algorithm.
- Discrete event-driven simulation.
- Network bandwidth management.
- Dijkstra's shortest-paths algorithm.
- ...

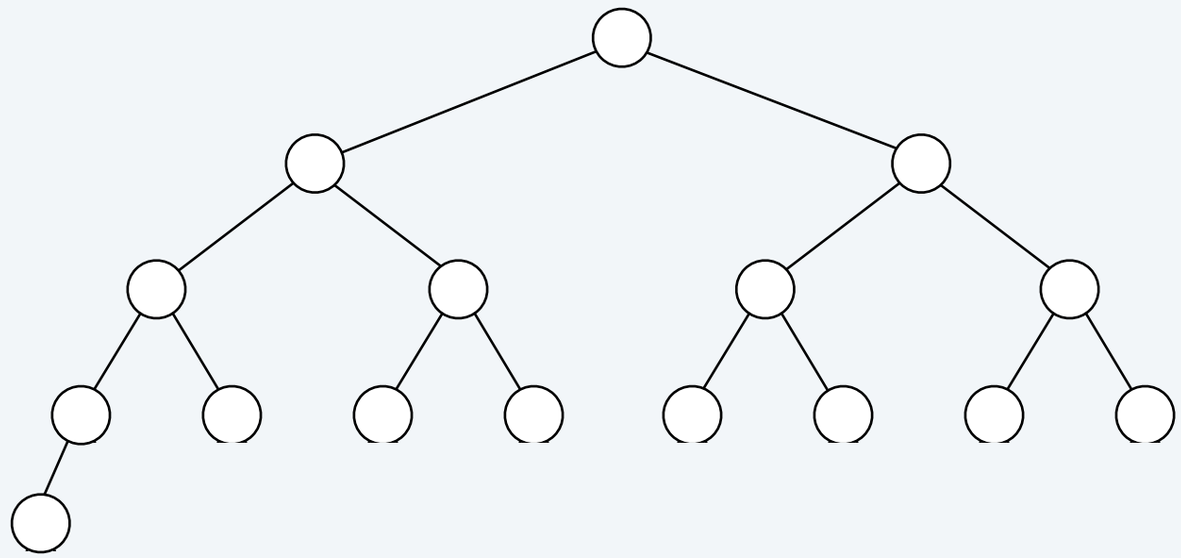


http://younginc.site11.com/source/5895/fos0092.html

Complete binary tree

Binary tree. Empty or node with links to two disjoint binary trees.

Complete tree. Perfectly balanced, except for bottom level.



complete tree with n = 16 nodes (height = 4)

Property. Height of complete binary tree with n nodes is $\lfloor \log_2 n \rfloor$.

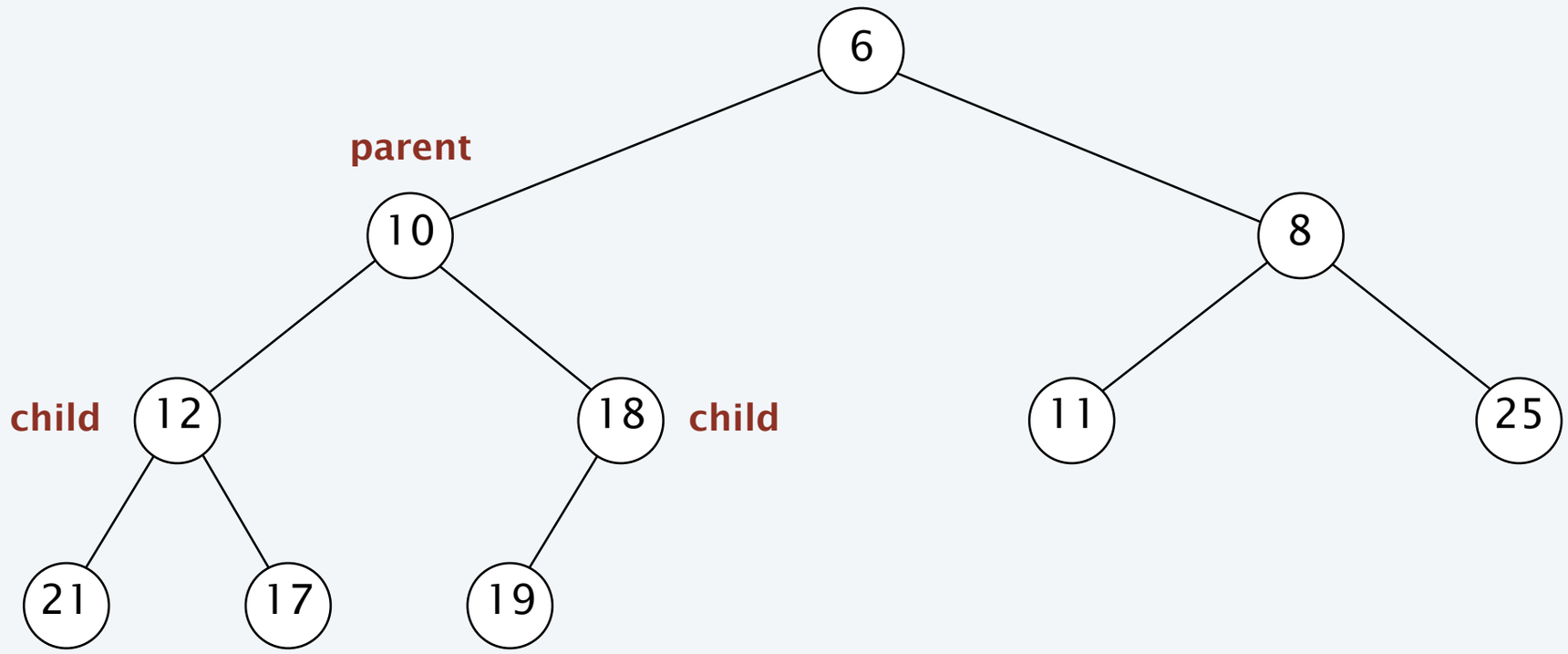
Pf. Height increases (by 1) only when n is a power of 2. ▪

A complete binary tree in nature

Binary heap

Binary heap. Heap-ordered complete binary tree.

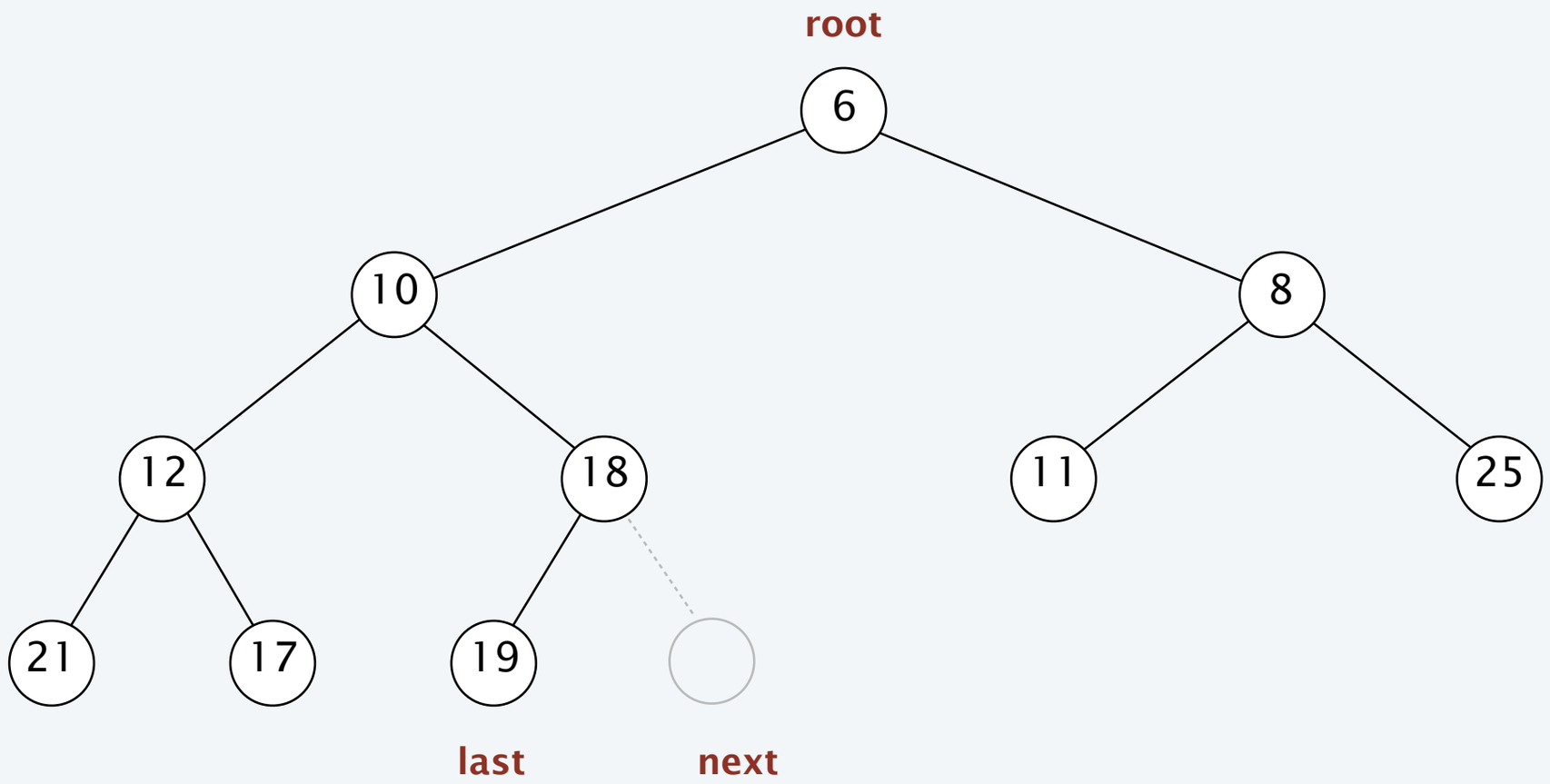
Heap-ordered tree. For each child, the key in child $\geq$ key in parent.



Explicit binary heap

Pointer representation. Each node has a pointer to parent and two children.

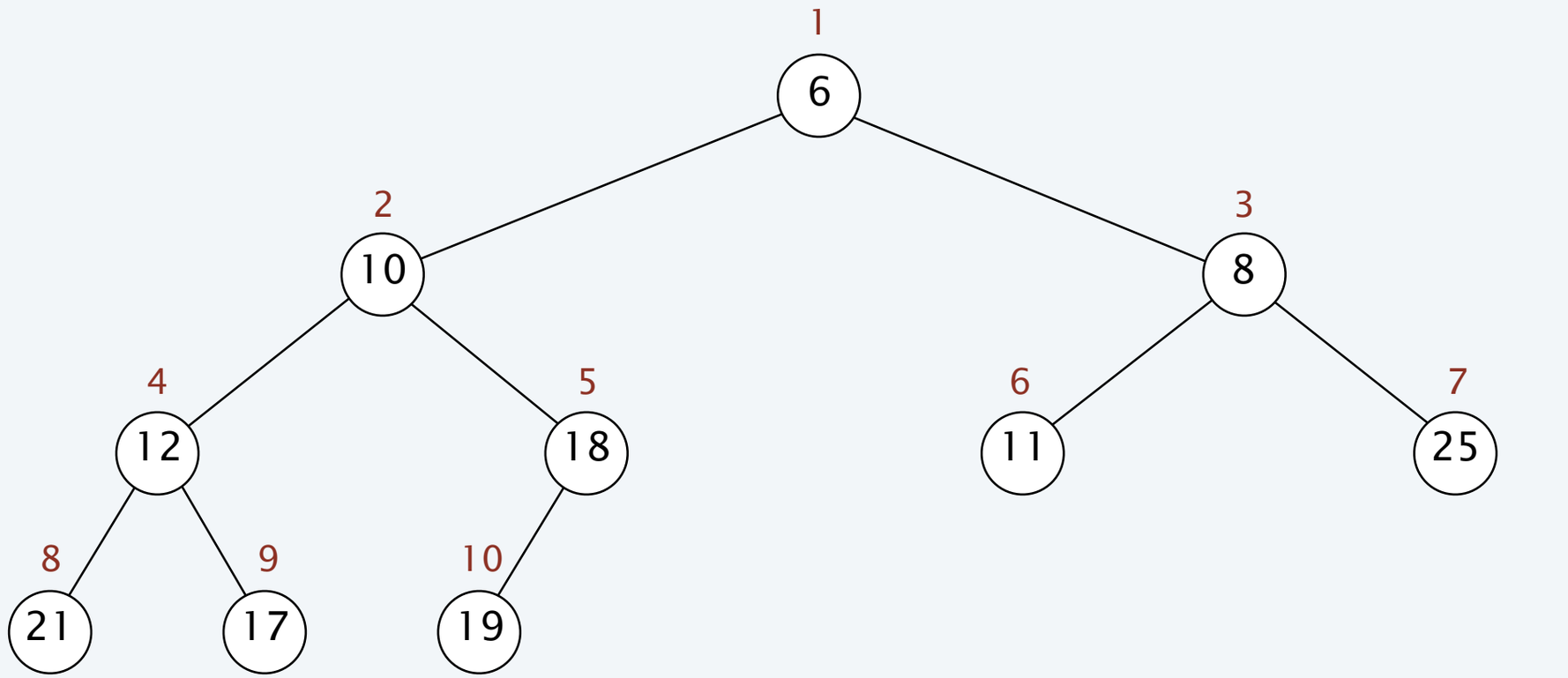
- Maintain number of elements n.
- Maintain pointer to root node.
- Can find pointer to last node or next node in $O(\log n)$ time.



Implicit binary heap

Array representation. Indices start at 1.

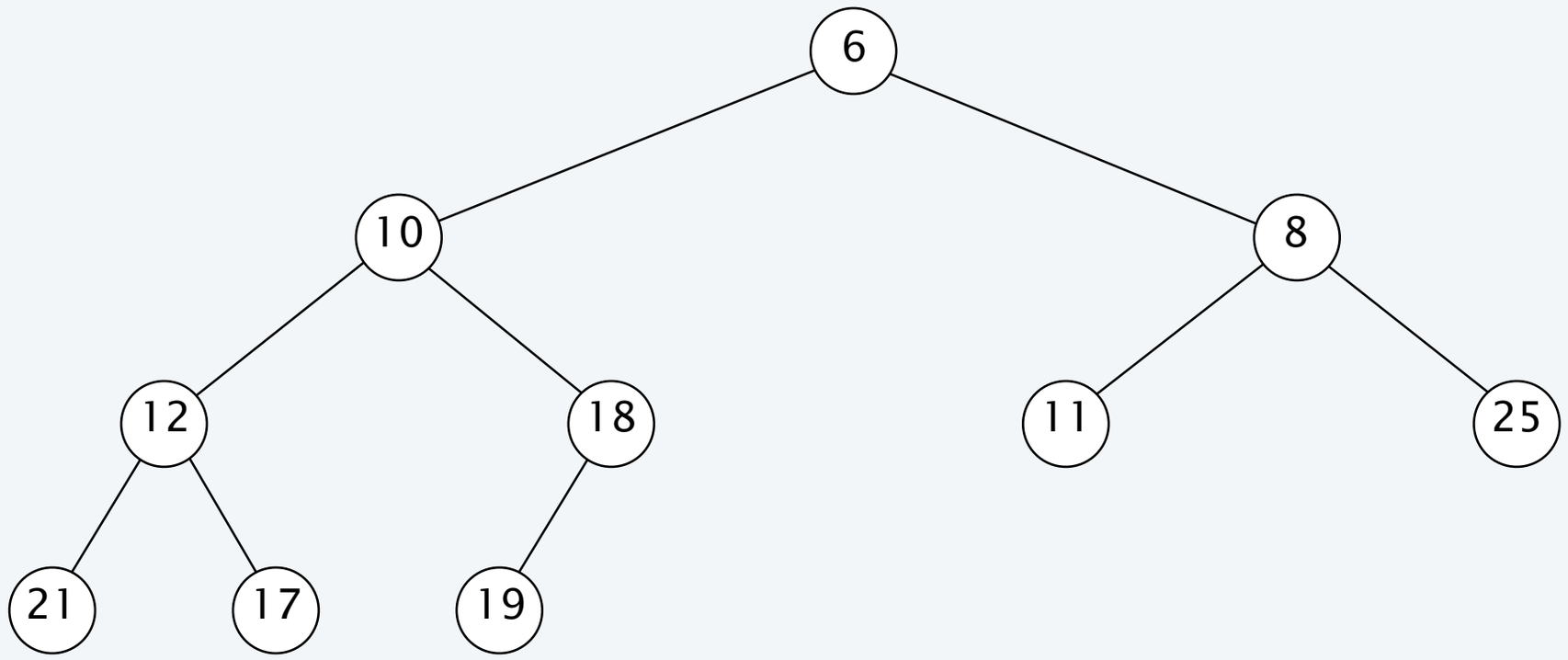
- Take nodes in level order.
- Parent of node at k is at $\lfloor k / 2 \rfloor$.
- Children of node at k are at $2k$ and $2k + 1$.



1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
6	10	8	12	18	11	25	21	17	19						

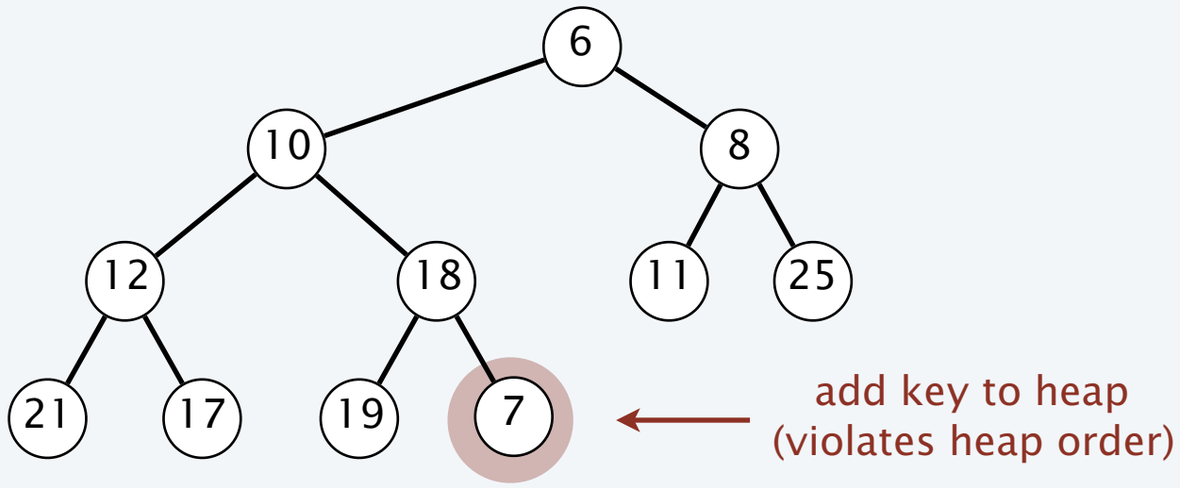
Binary heap demo

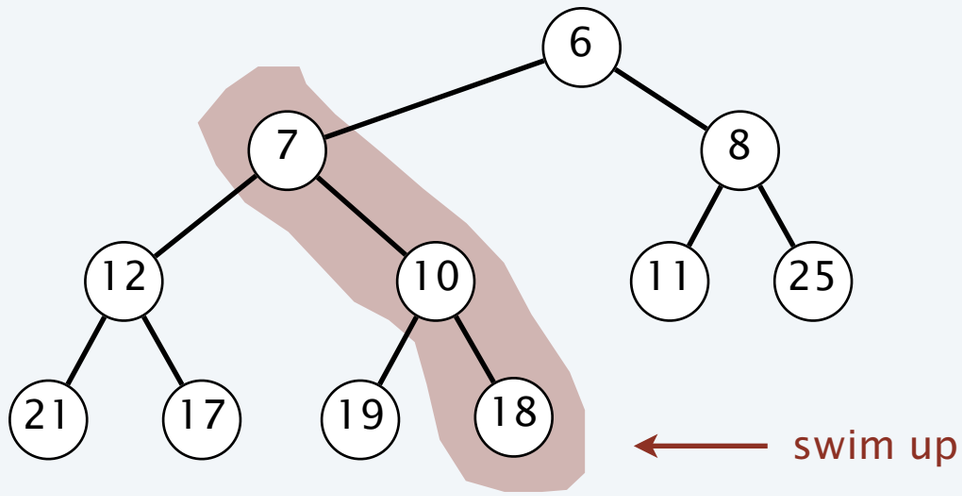
heap ordered



Binary heap: insert

Insert. Add element in new node at end; repeatedly exchange new element with element in its parent until heap order is restored.

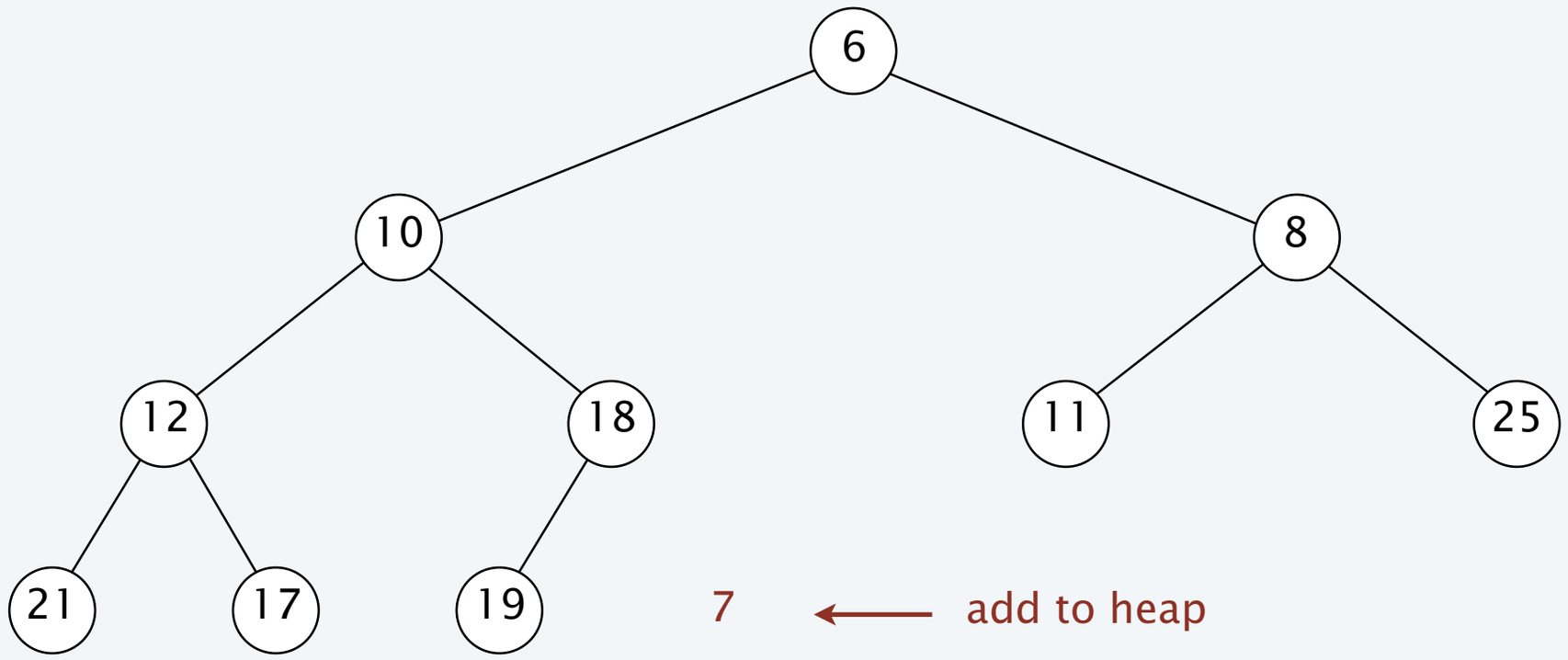




Binary heap demo

Insert. Add node at end; repeatedly exchange element in child with element in parent until heap order is restored.

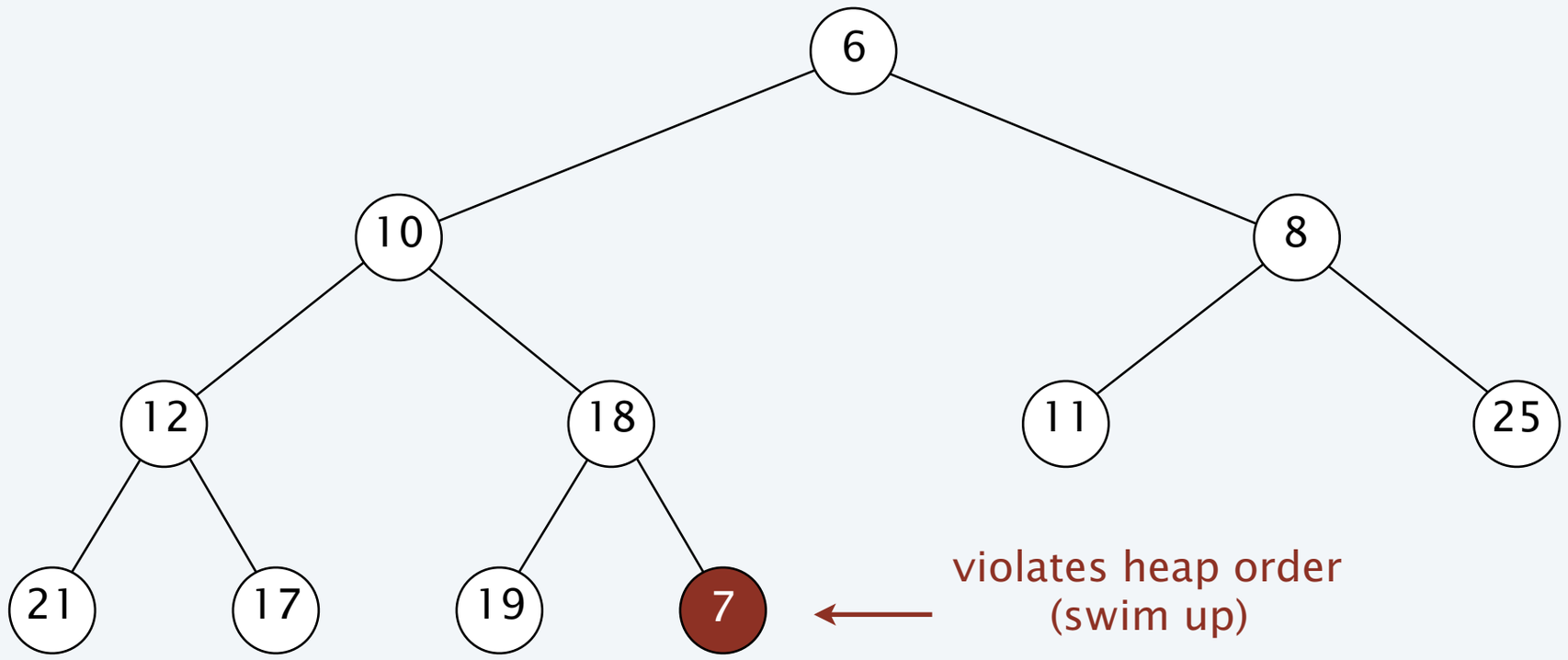
insert 7



Binary heap demo

Insert. Add node at end; repeatedly exchange element in child with element in parent until heap order is restored.

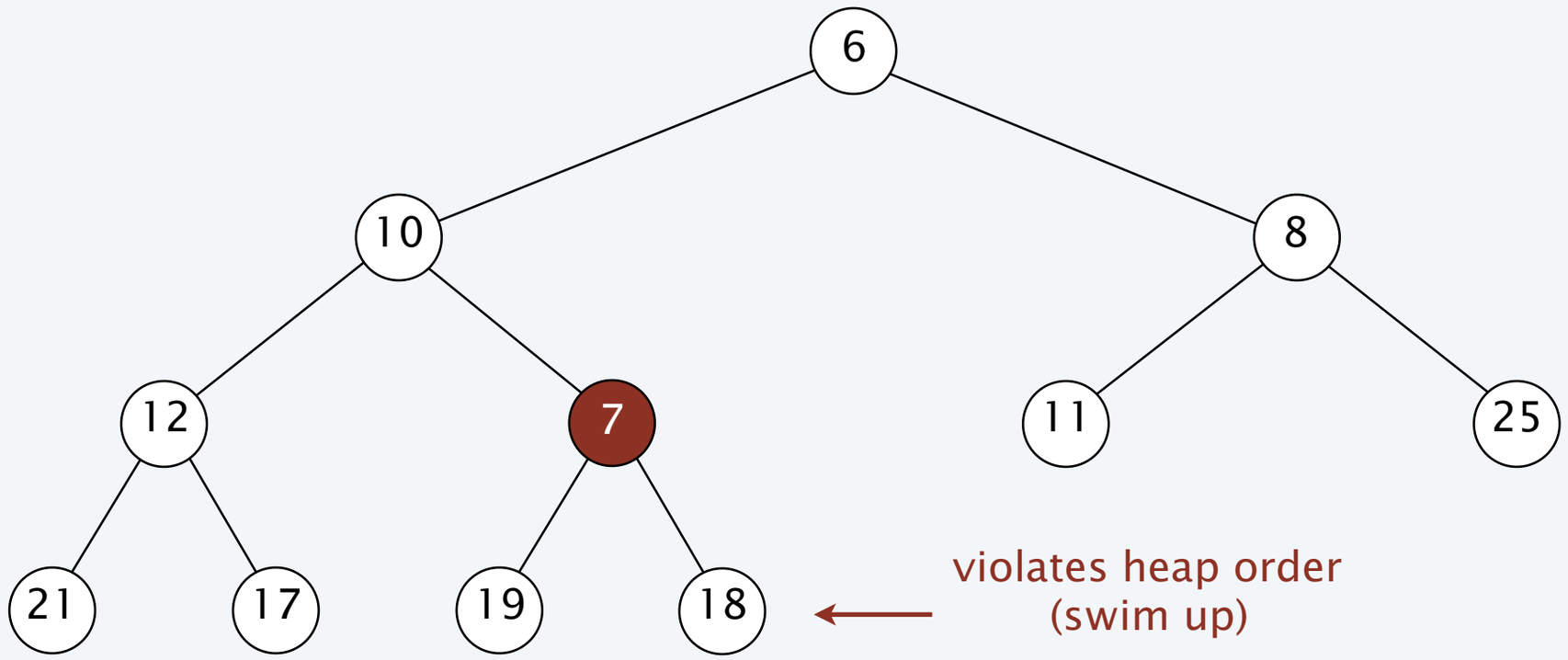
insert 7



Binary heap demo

Insert. Add node at end; repeatedly exchange element in child with element in parent until heap order is restored.

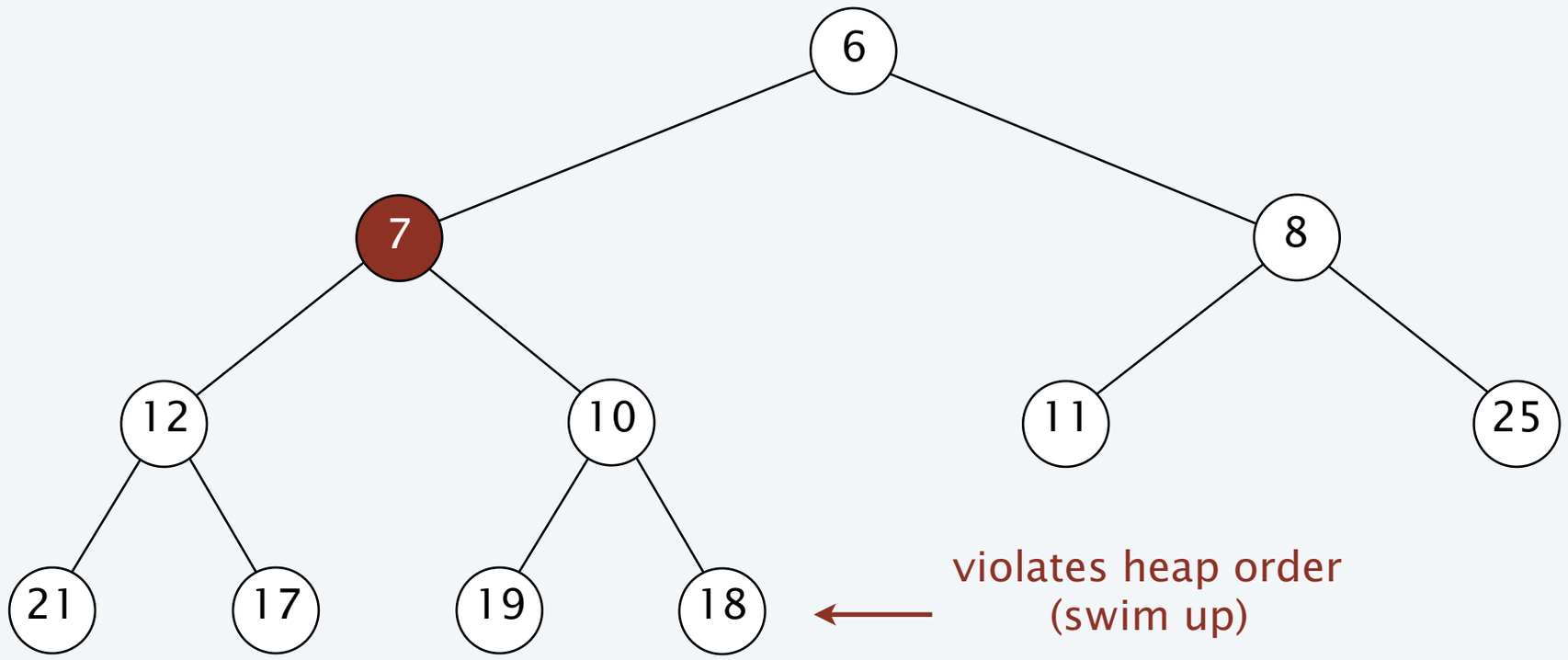
insert 7



Binary heap demo

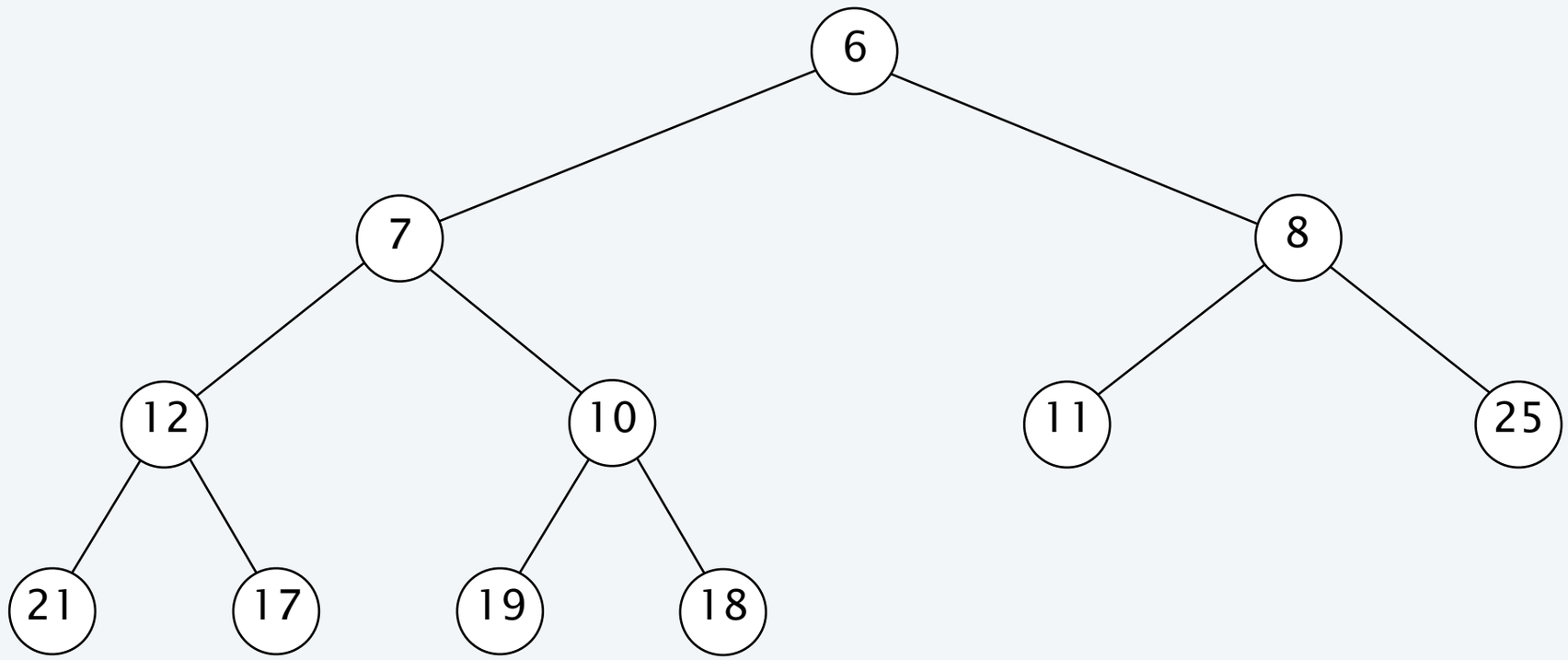
Insert. Add node at end; repeatedly exchange element in child with element in parent until heap order is restored.

insert 7



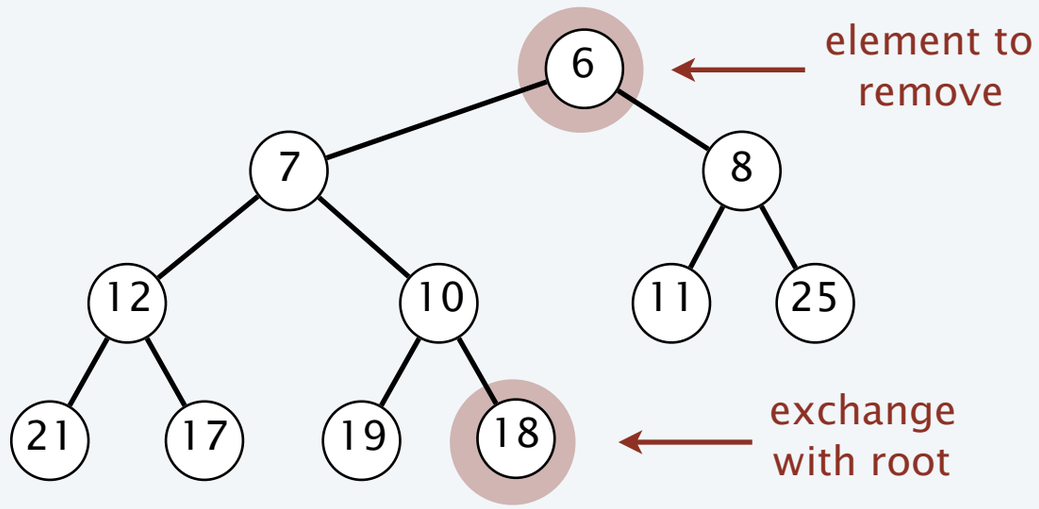
Binary heap demo

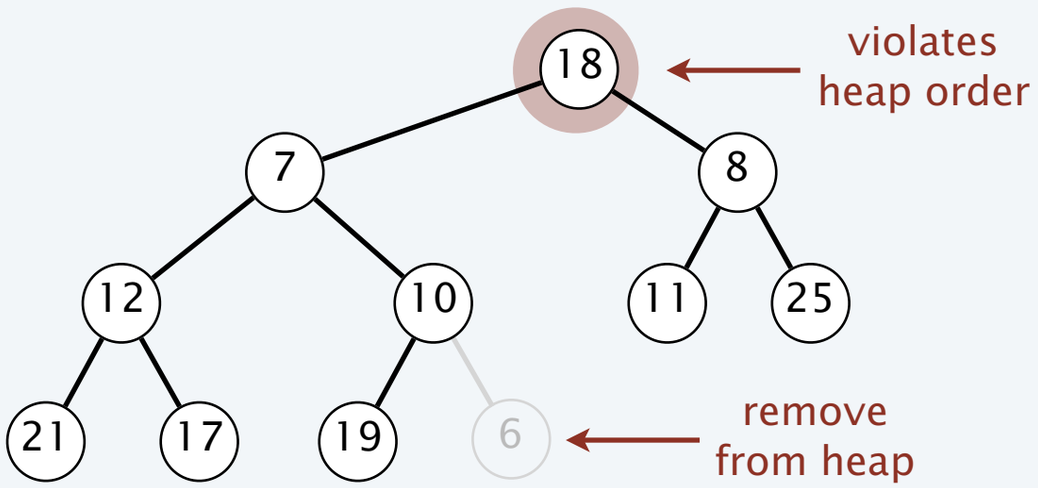
heap ordered

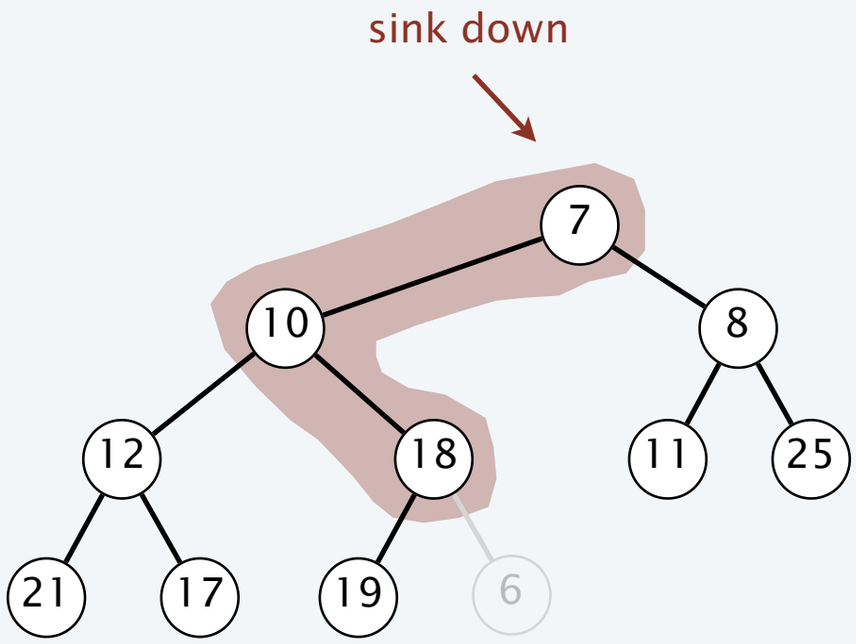


Binary heap: extract the minimum

Extract min. Exchange element in root node with last node; repeatedly exchange element in root with its smaller child until heap order is restored.



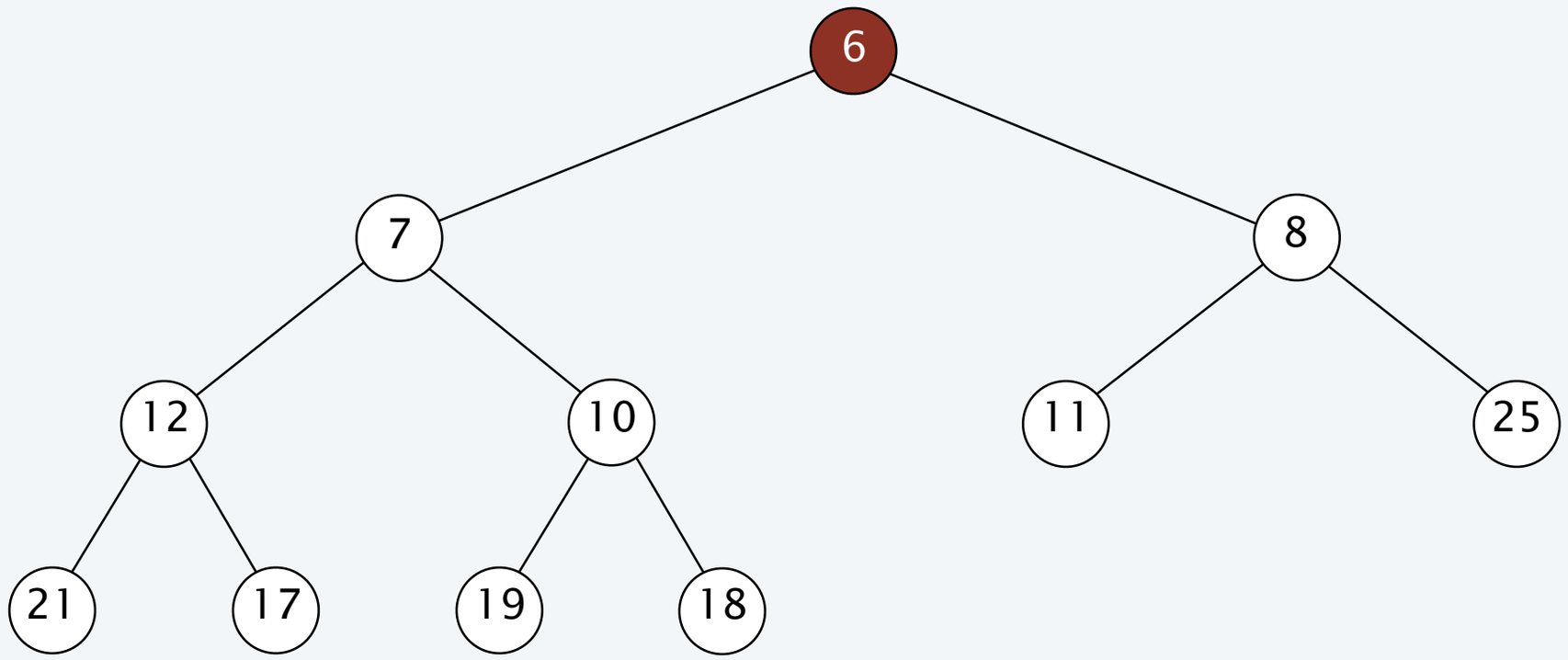




Binary heap demo

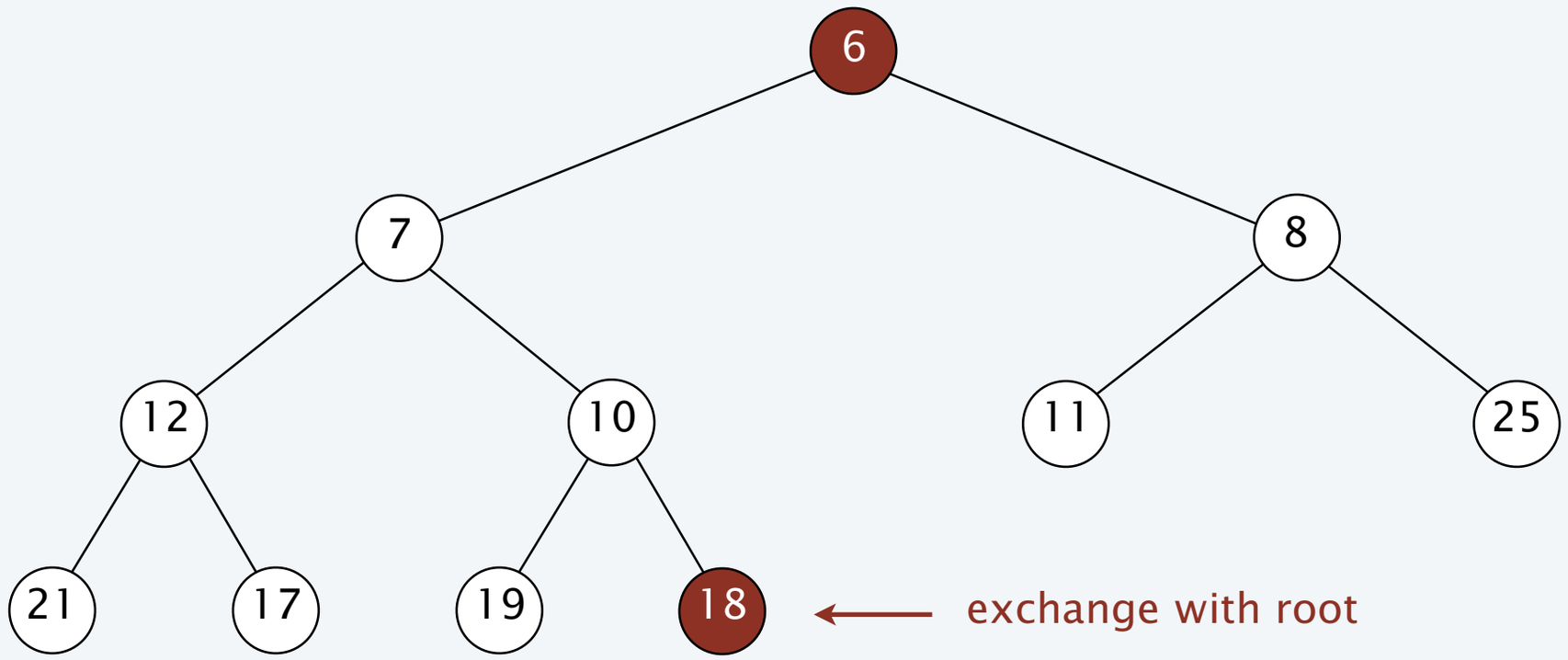
Extract min. Exchange root node with last node; repeatedly exchange element in parent with element in larger child until heap order is restored.

extract the minimum



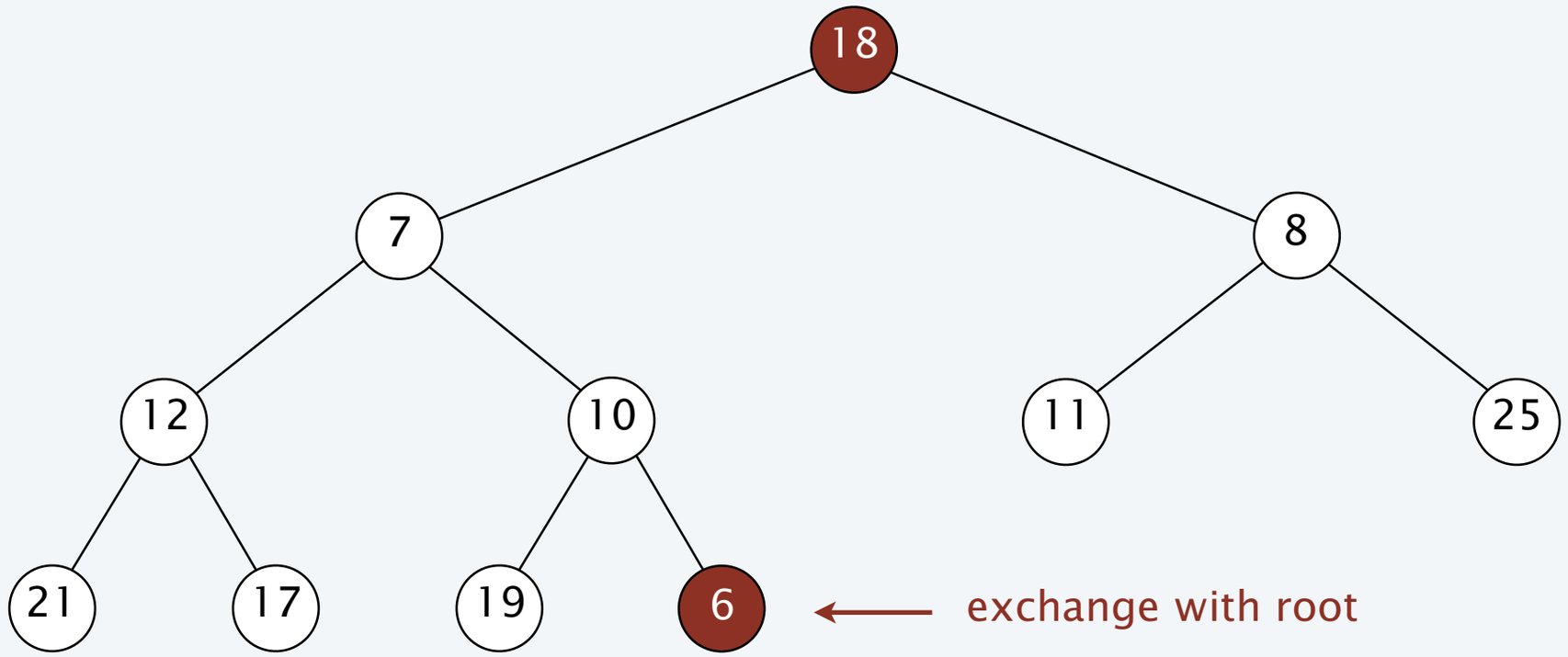
Binary heap demo

Extract min. Exchange root node with last node; repeatedly exchange element in parent with element in larger child until heap order is restored.



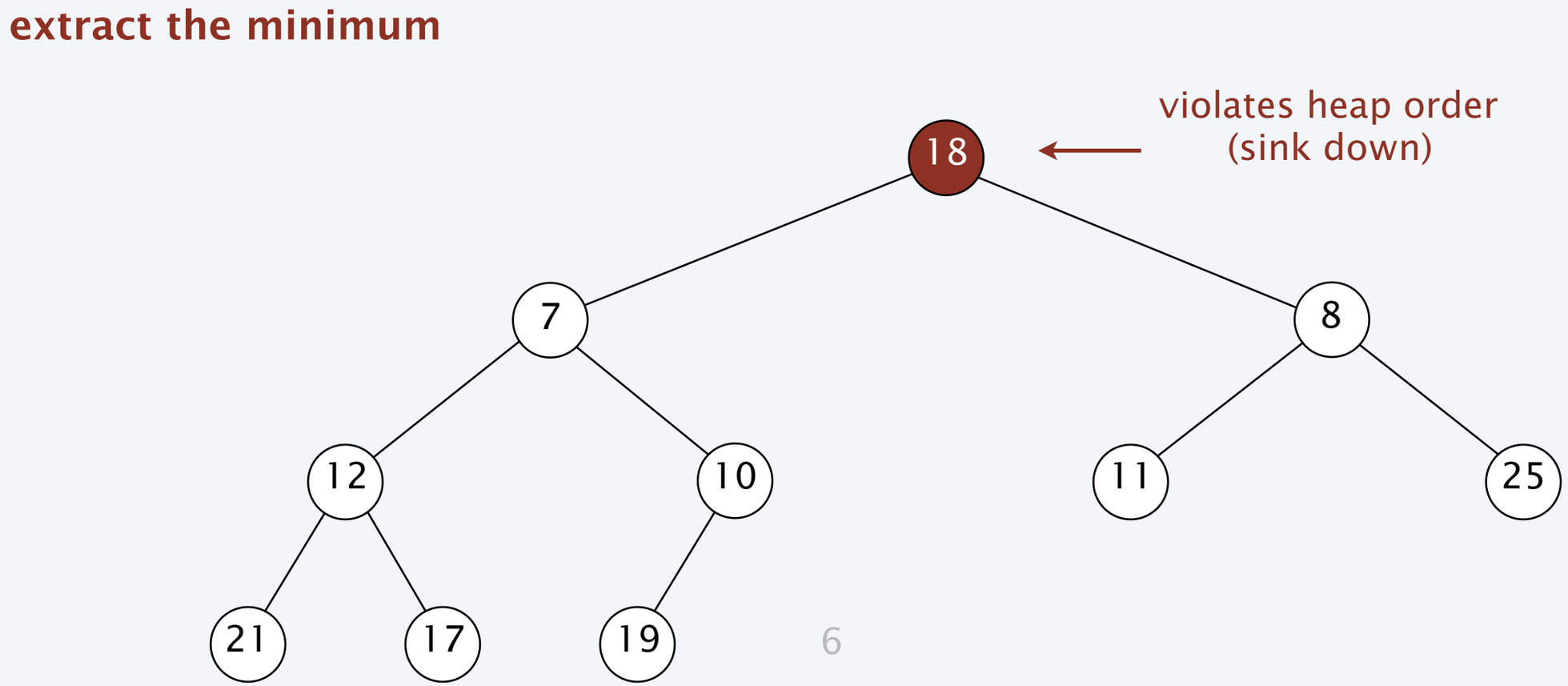
Binary heap demo

Extract min. Exchange root node with last node; repeatedly exchange element in parent with element in larger child until heap order is restored.



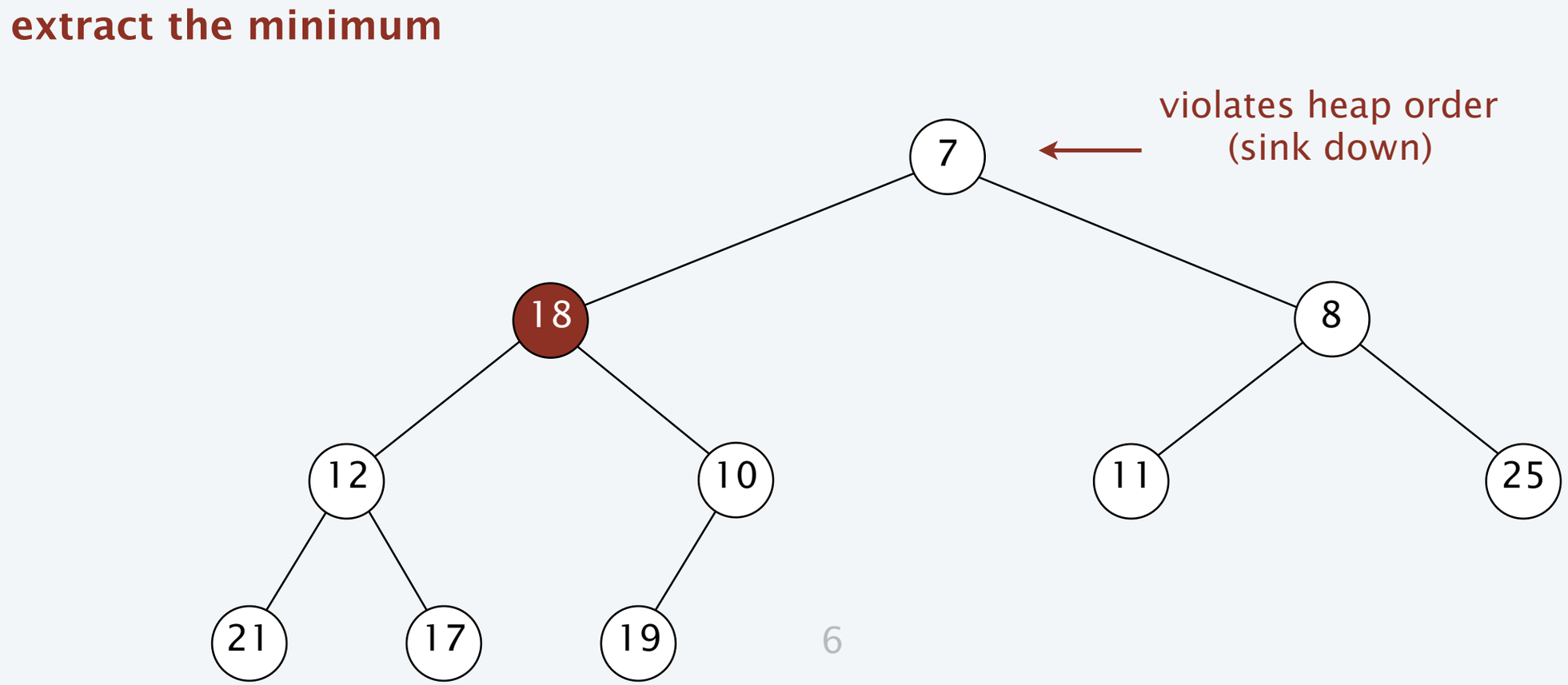
Binary heap demo

Extract min. Exchange root node with last node; repeatedly exchange element in parent with element in larger child until heap order is restored.



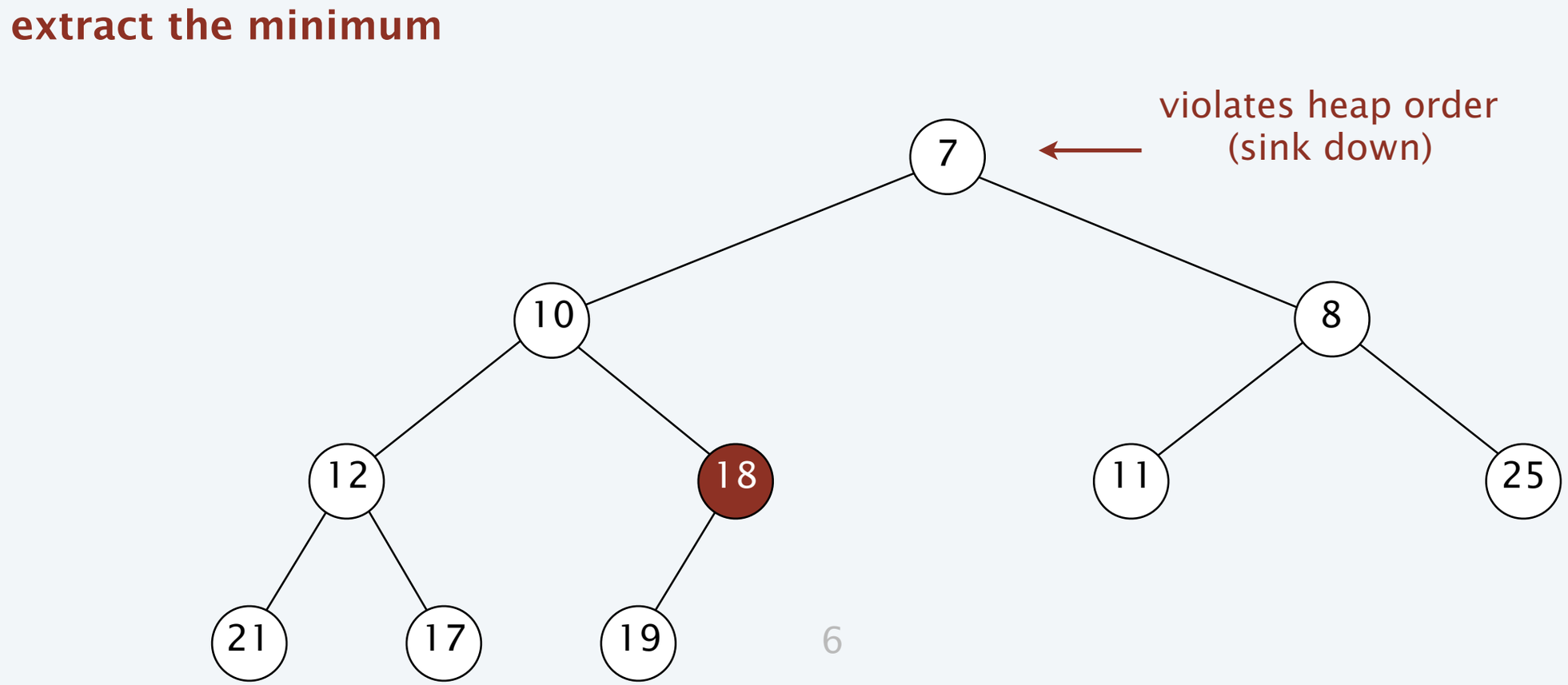
Binary heap demo

Extract min. Exchange root node with last node; repeatedly exchange element in parent with element in larger child until heap order is restored.



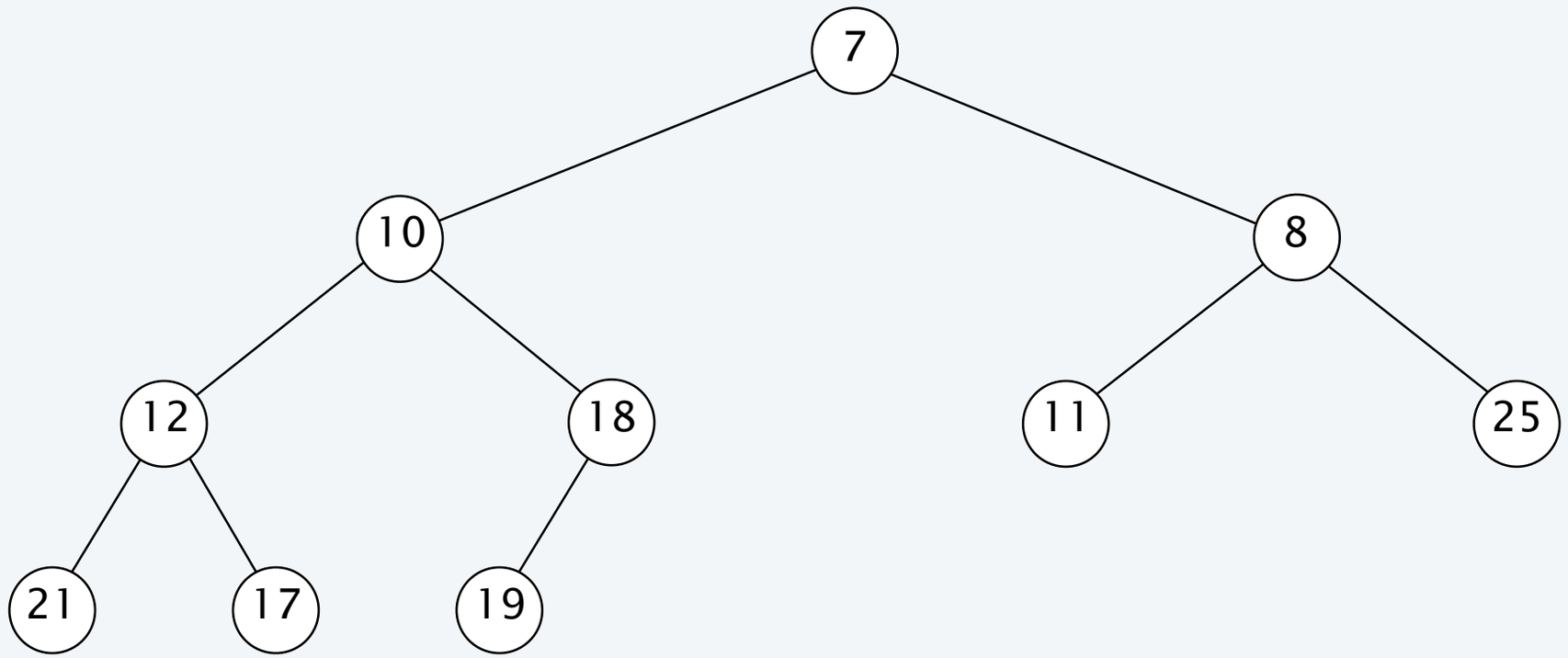
Binary heap demo

Extract min. Exchange root node with last node; repeatedly exchange element in parent with element in larger child until heap order is restored.



Binary heap demo

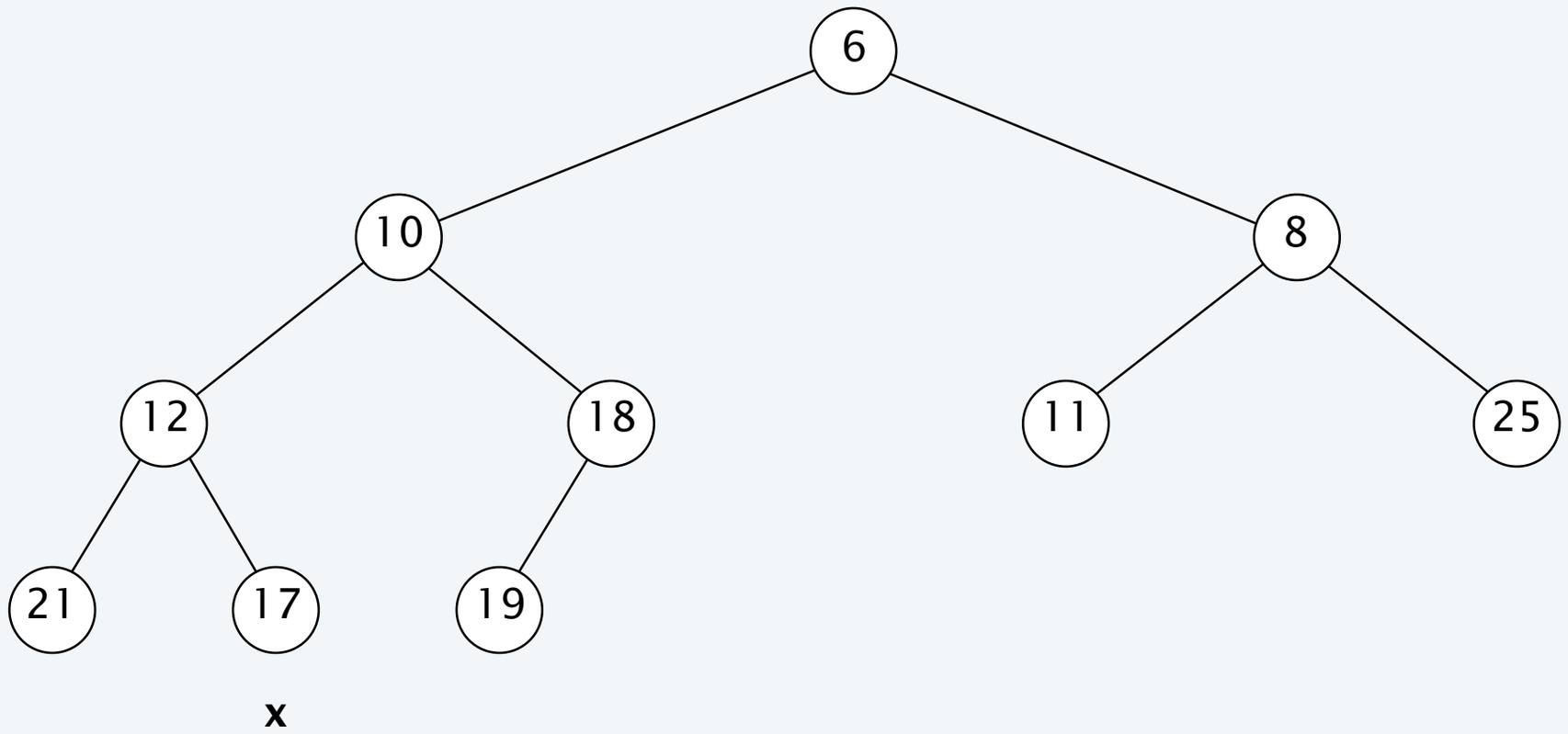
heap ordered



Binary heap: decrease key

Decrease key. Given a handle to node, repeatedly exchange element with its parent until heap order is restored.

decrease key of node x to 11



Binary heap: analysis

Theorem. In an implicit binary heap, any sequence of m INSERT, EXTRACT-MIN, and DECREASE-KEY operations with n INSERT operations takes $O(m \log n)$ time.

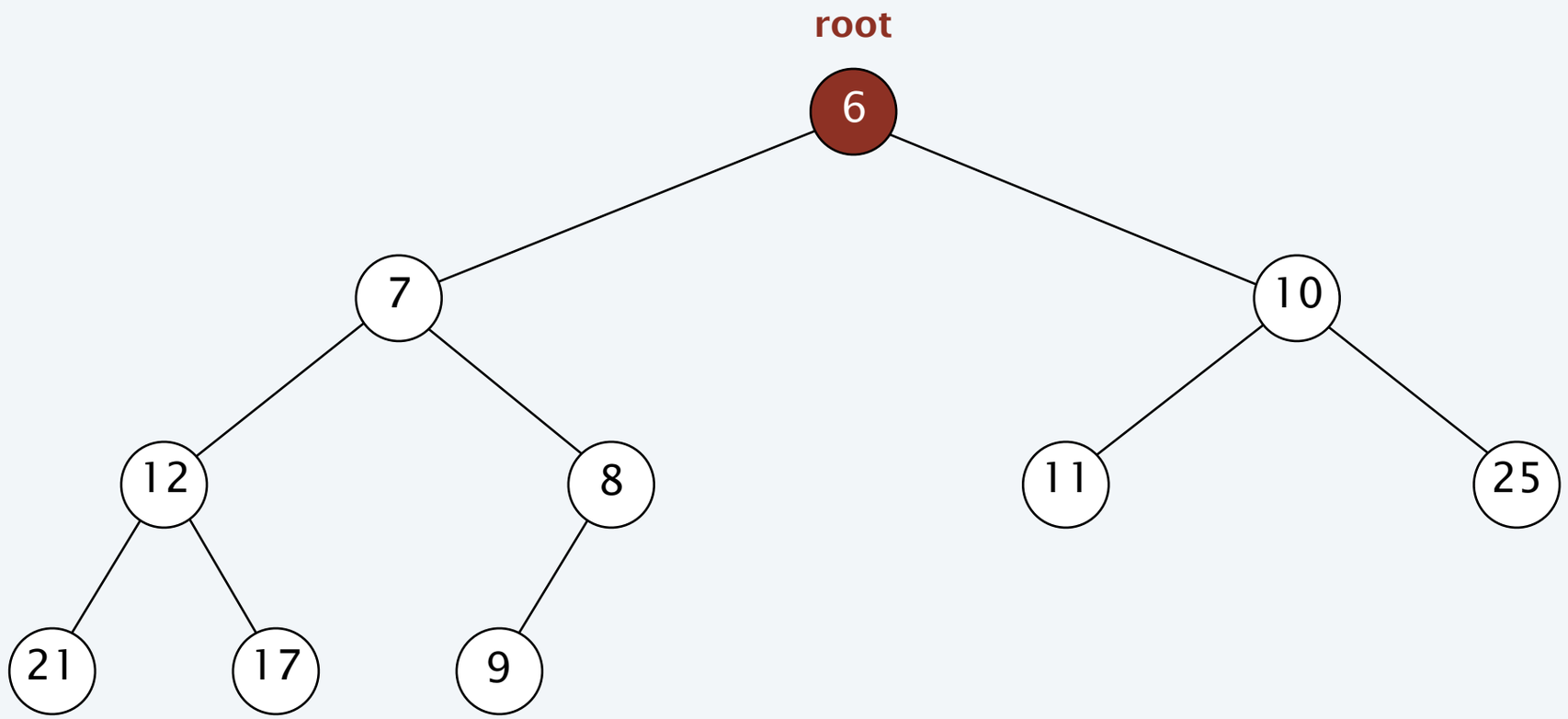
Pf.

- Each heap op touches nodes only on a path from the root to a leaf; the height of the tree is at most $\log_2 n$.
- The total cost of expanding and contracting the arrays is $O(n)$. ▪

Theorem. In an explicit binary heap with n nodes, the operations INSERT, DECREASE-KEY, and EXTRACT-MIN take $O(\log n)$ time in the worst case.

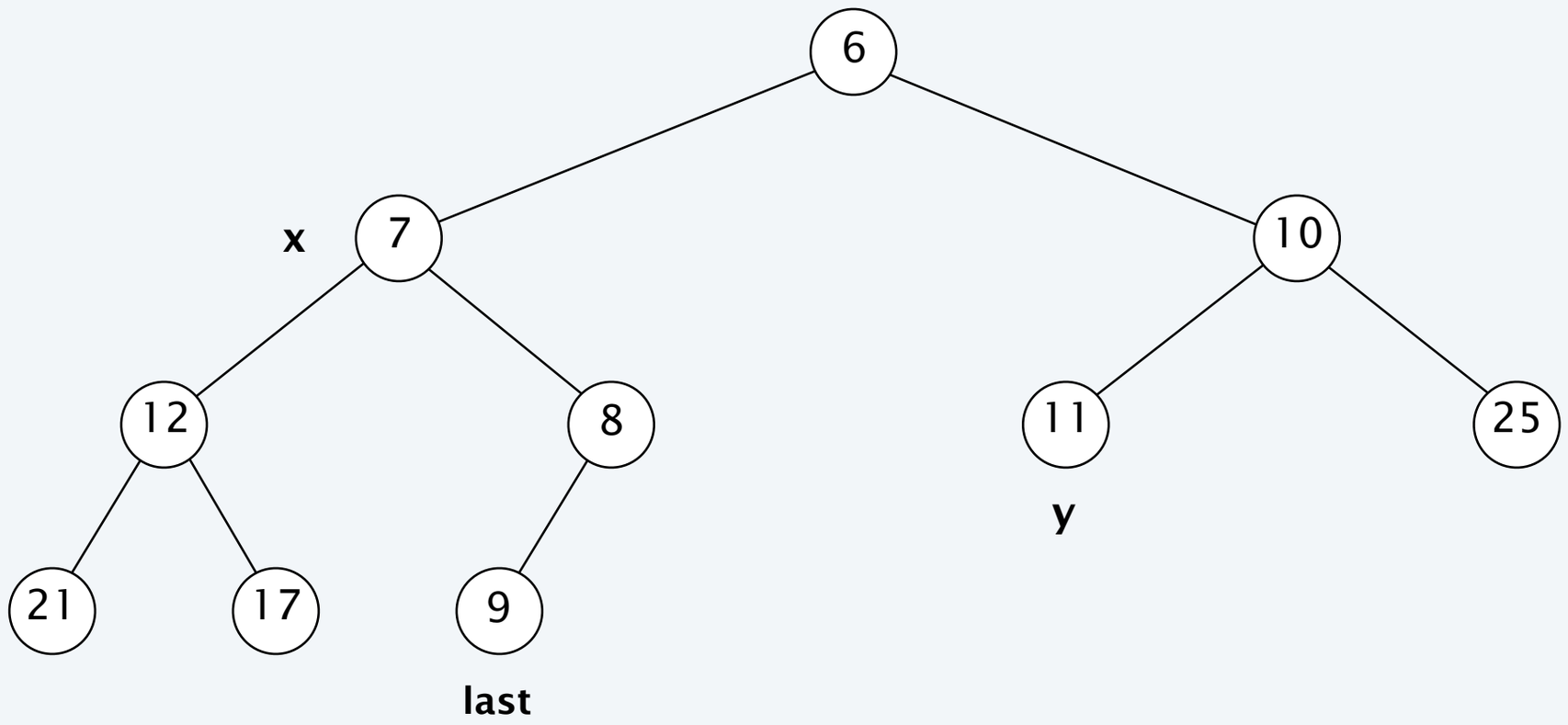
Binary heap: find-min

Find the minimum. Return element in the root node.



Binary heap: delete

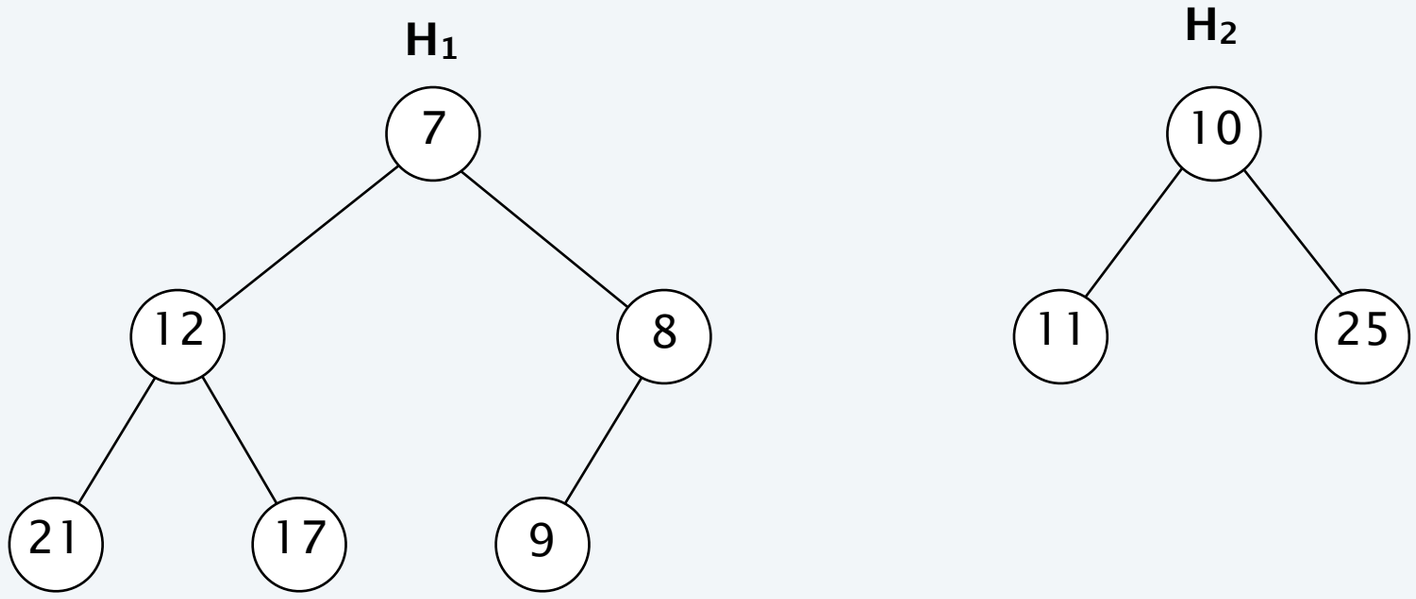
Delete. Given a handle to a node, exchange element in node with last node; either swim down or sink up the node until heap order is restored.



Binary heap: meld

Meld. Given two binary heaps H_1 and H_2, merge into a single binary heap.

Observation. No easy solution: $\Omega(n)$ time apparently required.

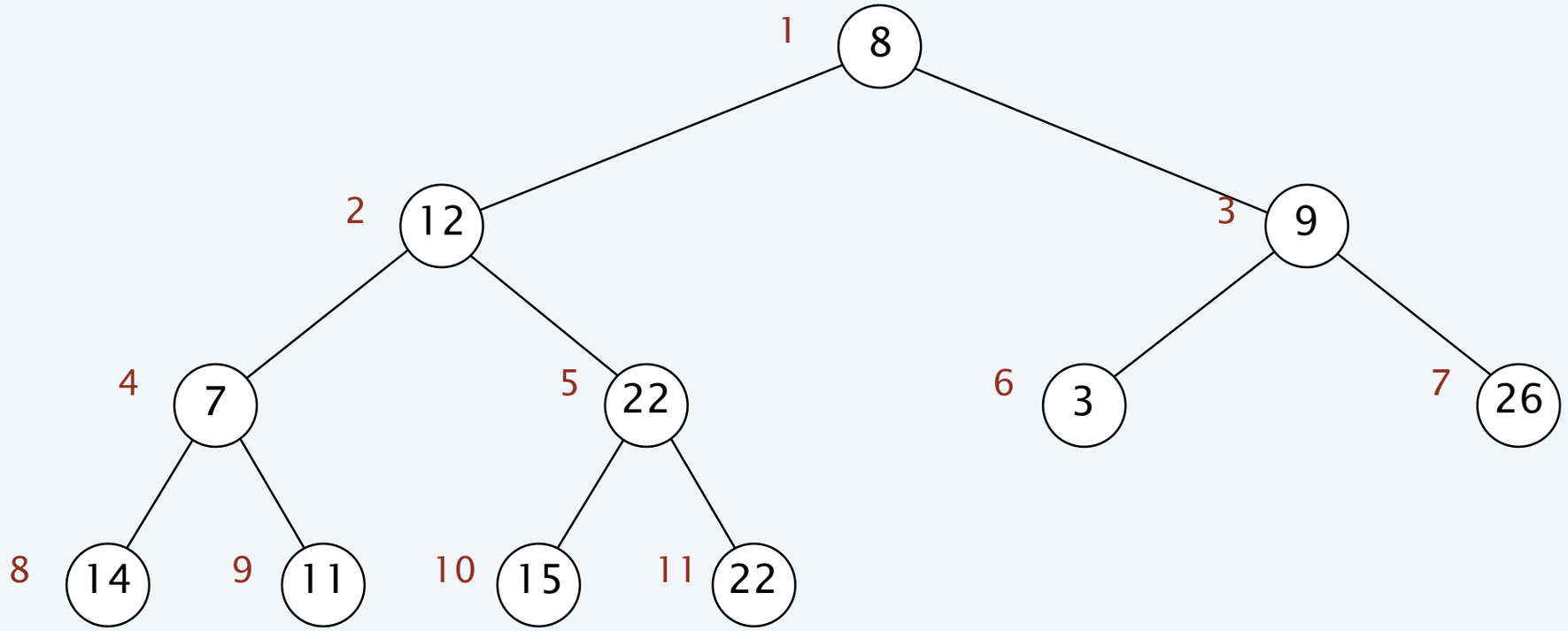


Binary heap: heapify

Heapify. Given n elements, construct a binary heap containing them.

Observation. Can do in $O(n \log n)$ time by inserting each element.

Bottom-up method. For $i = n$ to 1, repeatedly exchange the element in node i with its smaller child until subtree rooted at i is heap-ordered.



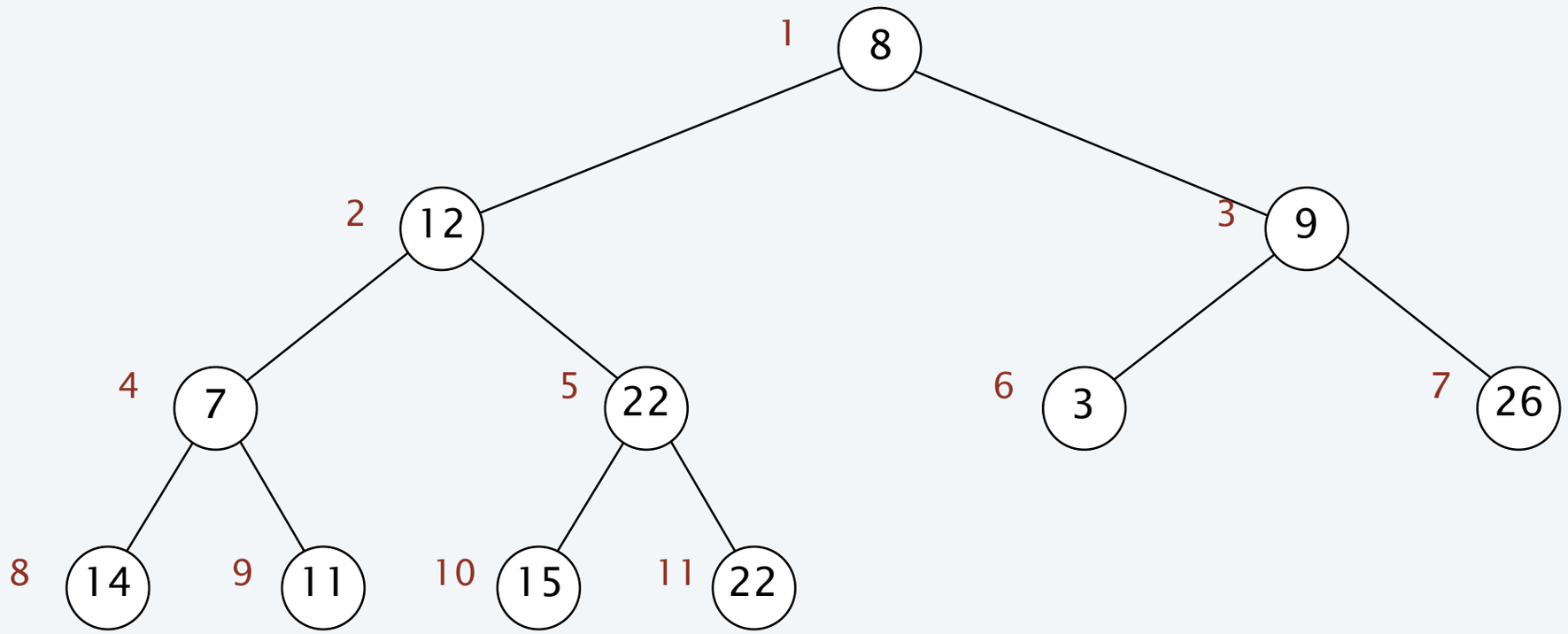
8	12	9	7	22	3	26	14	11	15	22
1	2	3	4	5	6	7	8	9	10	11

Heapify demo

Heapify. For each element in reverse-array order, sink it down.

we assume array entries are indexed 1 to n

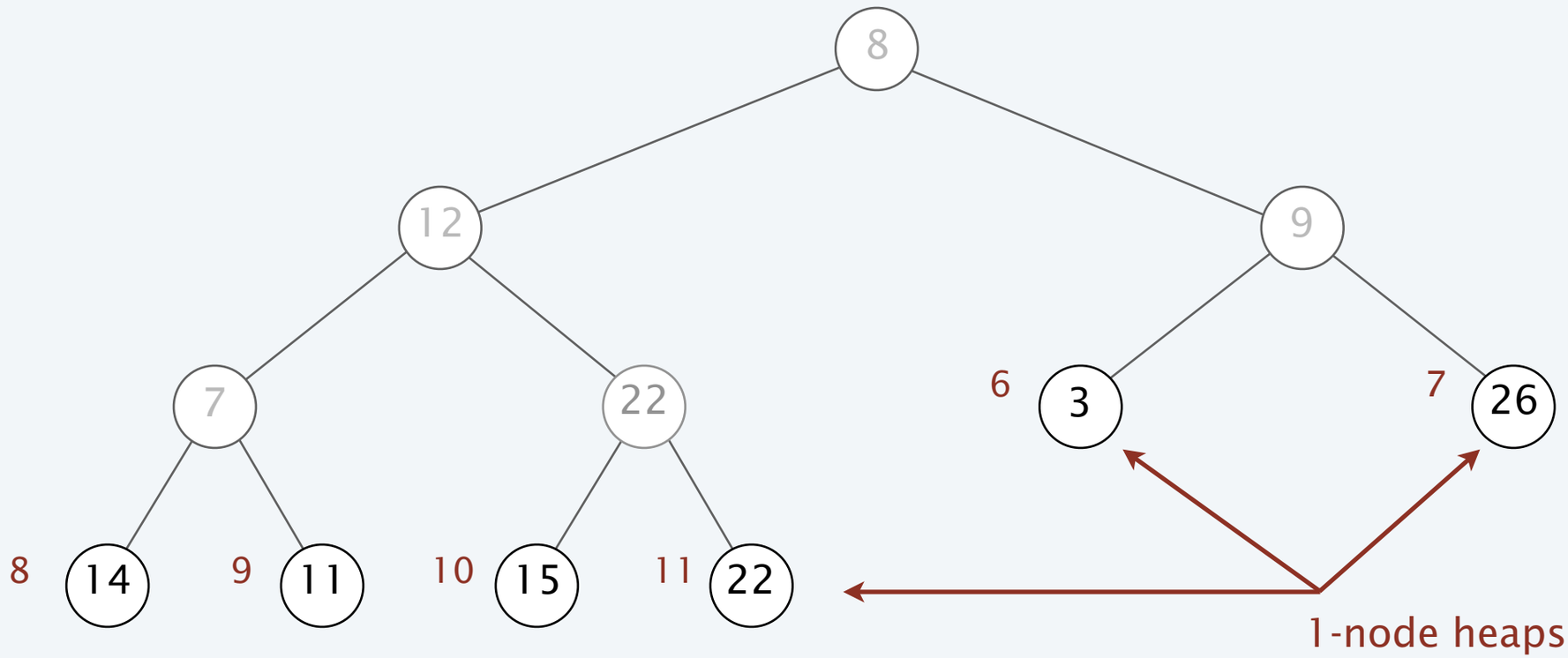
array in arbitrary order



8	12	9	7	22	3	26	14	11	15	22
1	2	3	4	5	6	7	8	9	10	11

Heapify demo

Heapify. For each element in reverse-array order, sink it down.

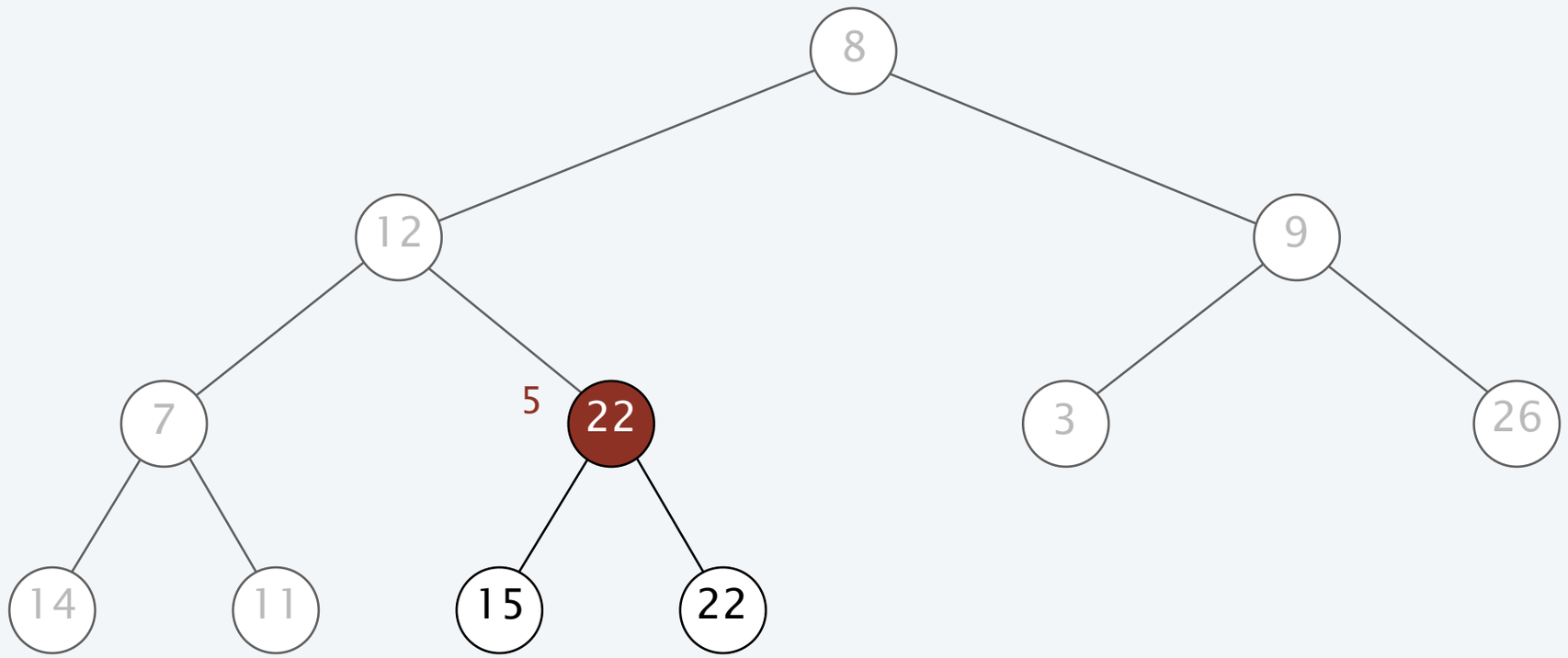


8	12	9	7	22	3	26	14	11	15	22
					6	7	8	9	10	11

Heapify demo

Heapify. For each element in reverse-array order, sink it down.

sink 5

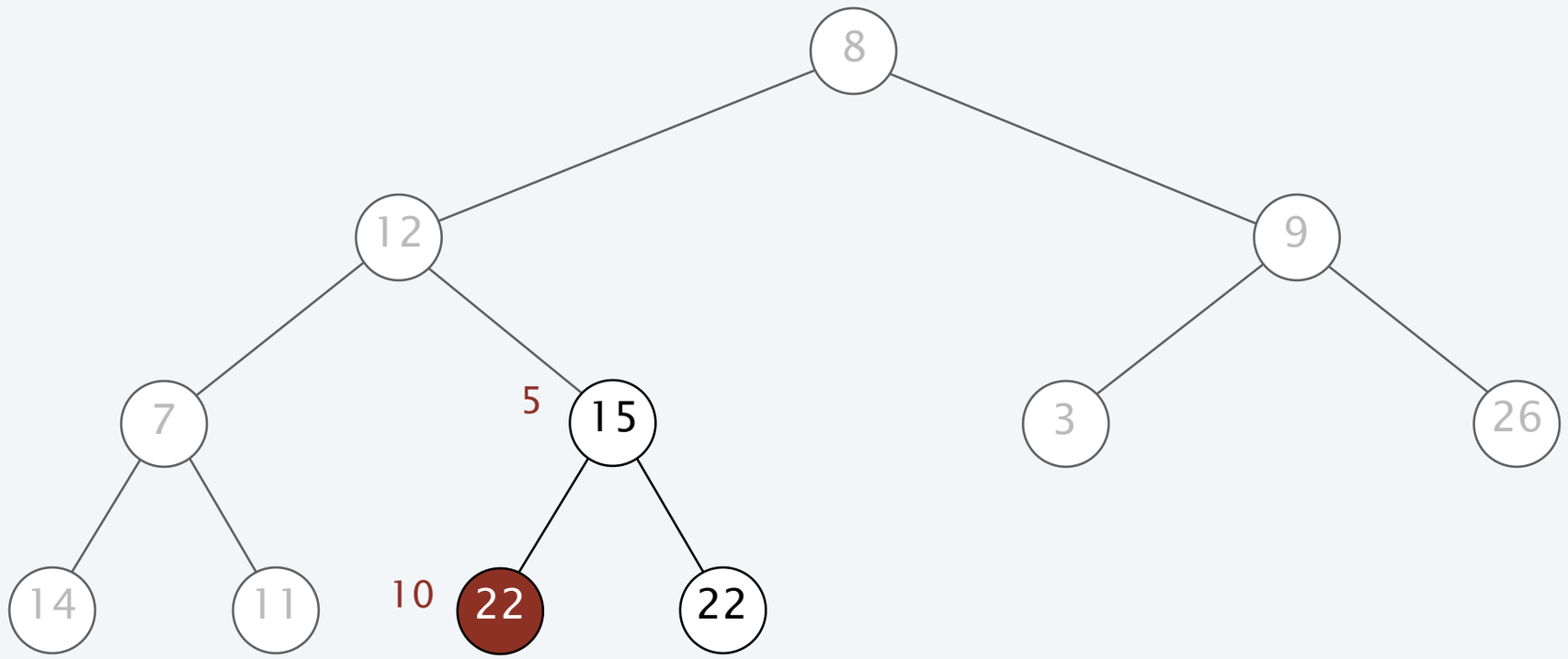


8	12	9	7	22	3	26	14	11	15	22
				5						

Heapify demo

Heapify. For each element in reverse-array order, sink it down.

sink 5

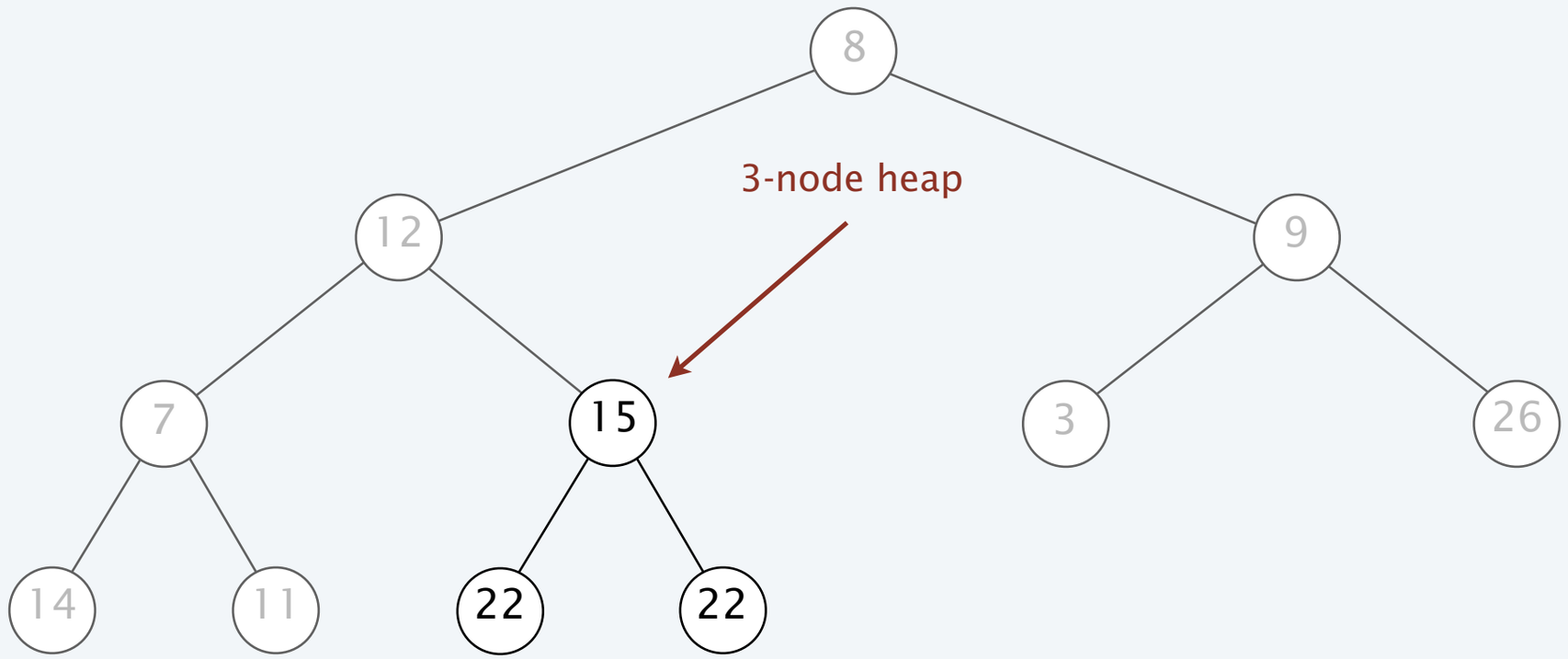


8	12	9	7	15	3	26	14	11	22	22
				5					10	

Heapify demo

Heapify. For each element in reverse-array order, sink it down.

sink 5

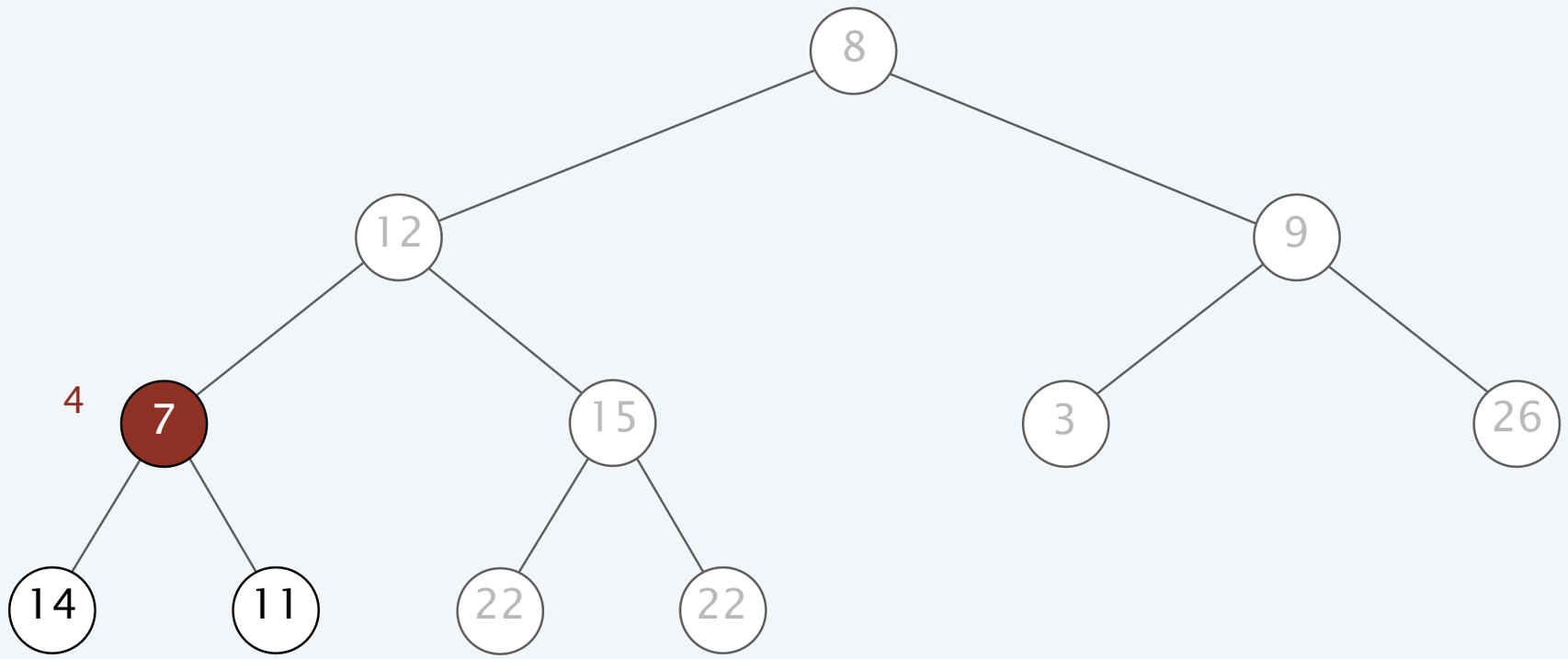


8	12	9	7	15	3	26	14	11	22	22

Heapify demo

Heapify. For each element in reverse-array order, sink it down.

sink 4

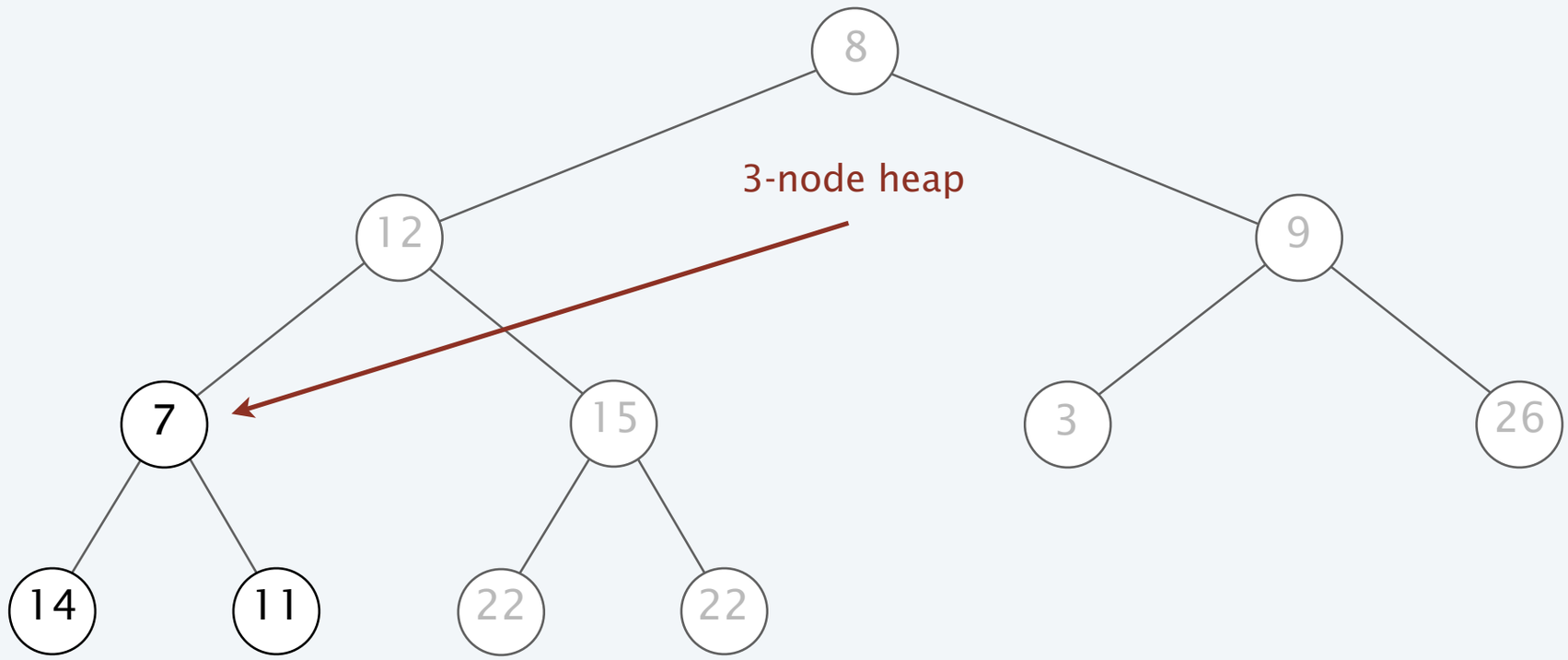


8	12	9	7	15	3	26	14	11	22	22
			4							

Heapify demo

Heapify. For each element in reverse-array order, sink it down.

sink 4

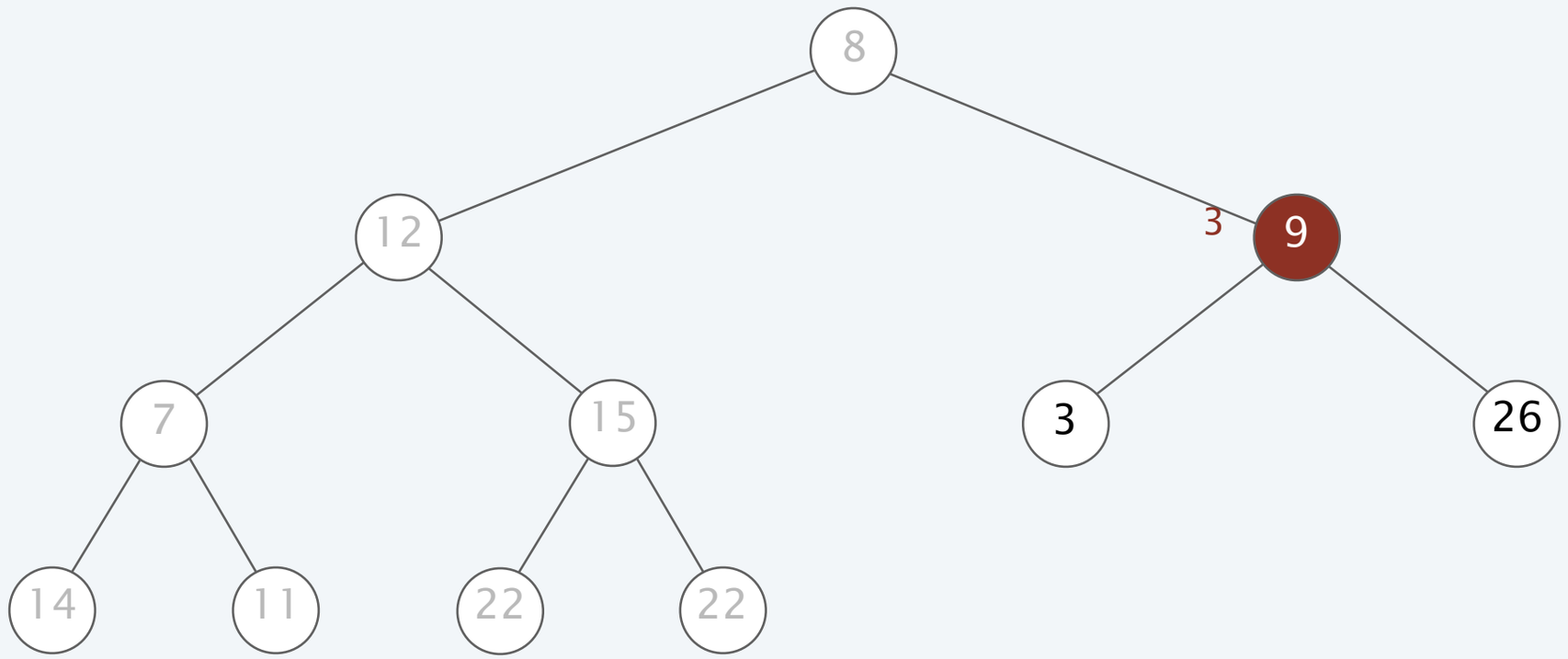


8	12	9	7	15	3	26	14	11	22	22

Heapify demo

Heapify. For each element in reverse-array order, sink it down.

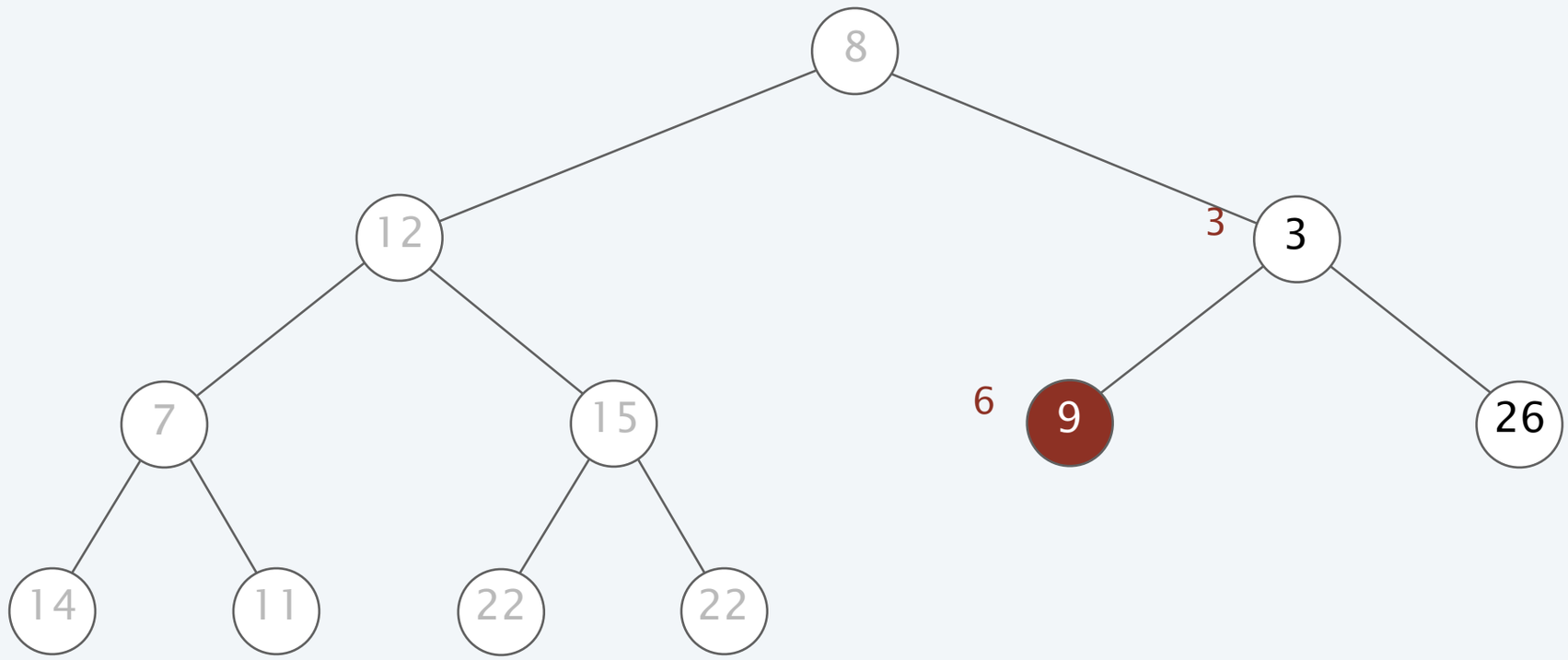
sink 3



Heapify demo

Heapify. For each element in reverse-array order, sink it down.

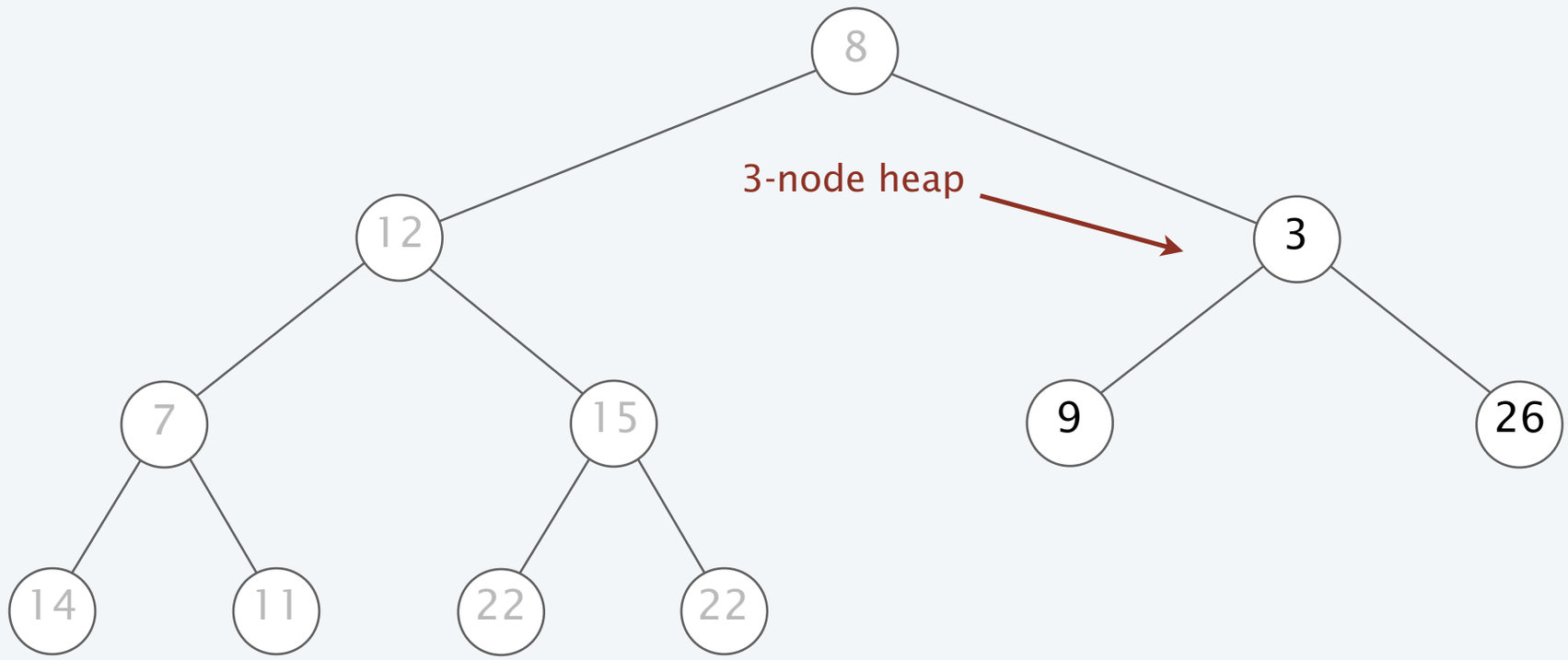
sink 3



Heapify demo

Heapify. For each element in reverse-array order, sink it down.

sink 3

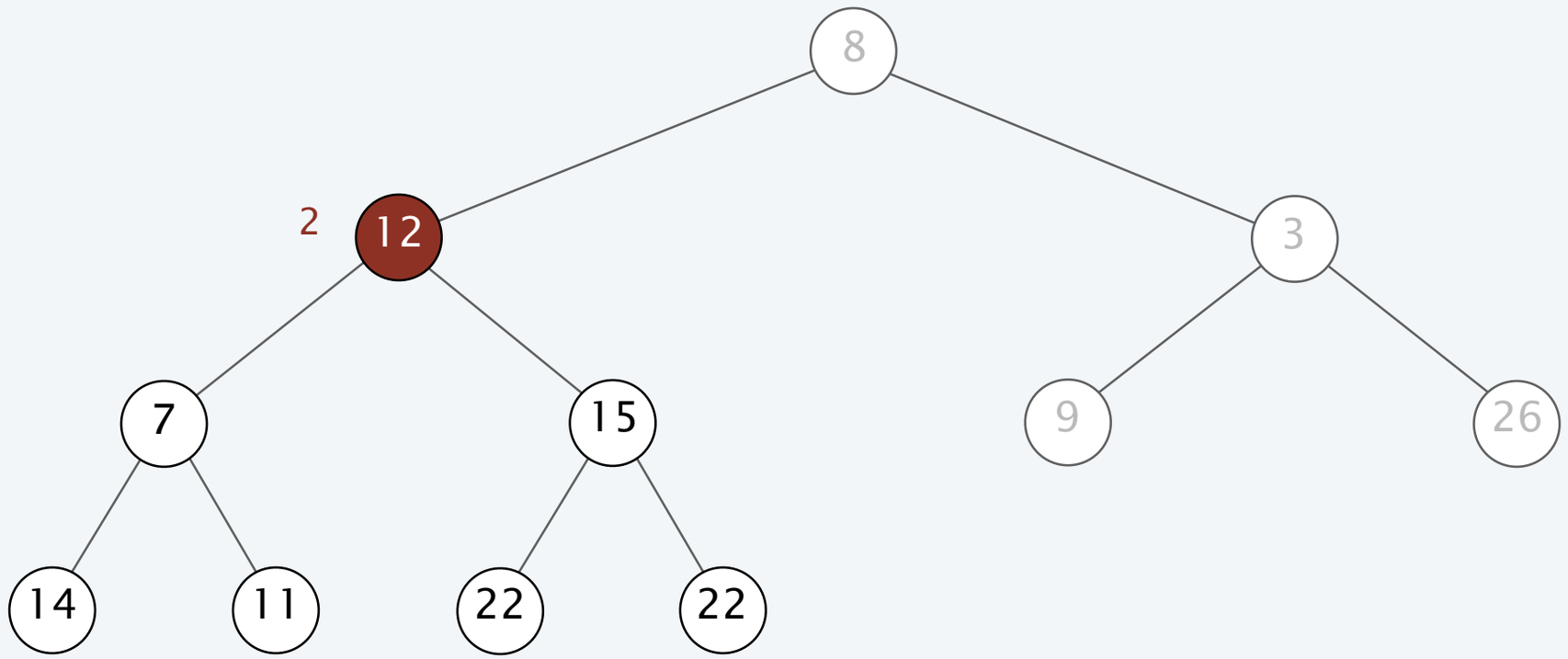


8	12	3	7	15	26	26	14	11	22	22

Heapify demo

Heapify. For each element in reverse-array order, sink it down.

sink 2

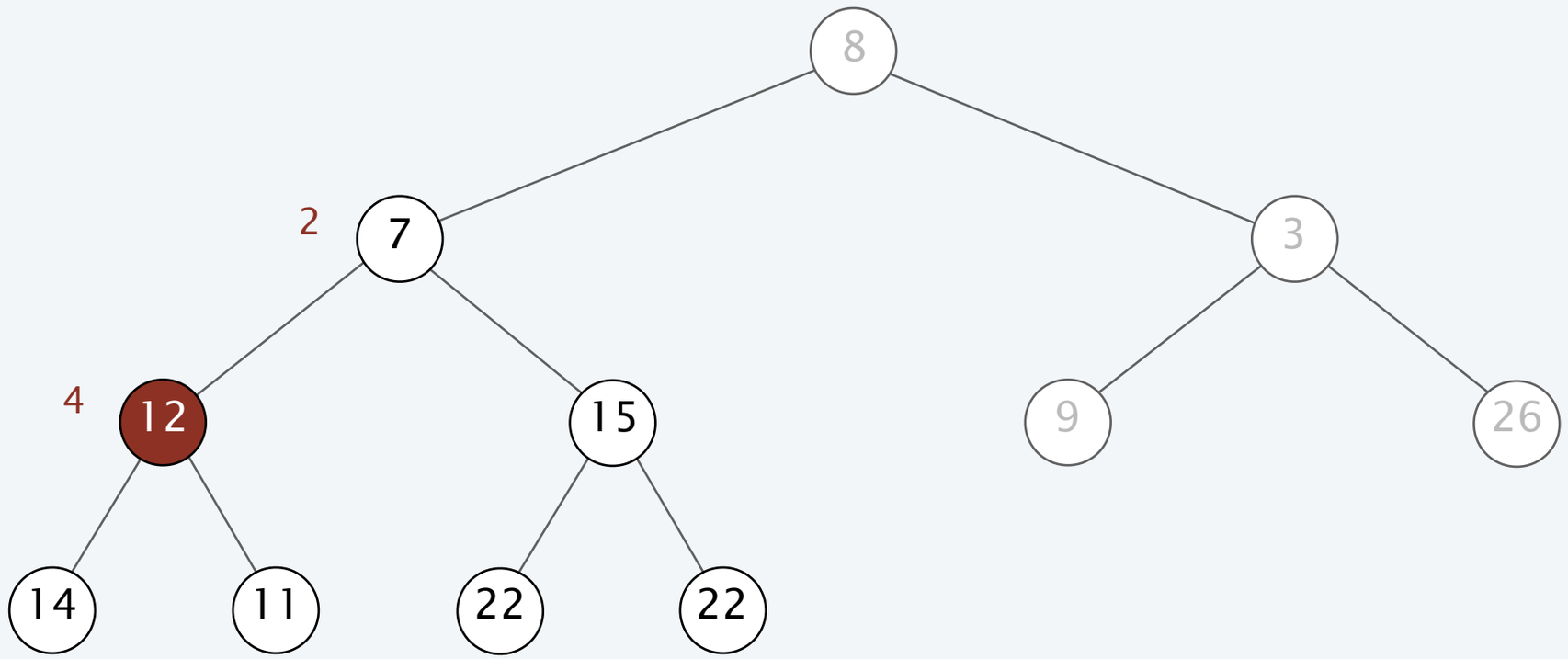


8	12	3	7	15	9	26	14	11	22	22
	2									

Heapify demo

Heapify. For each element in reverse-array order, sink it down.

sink 2

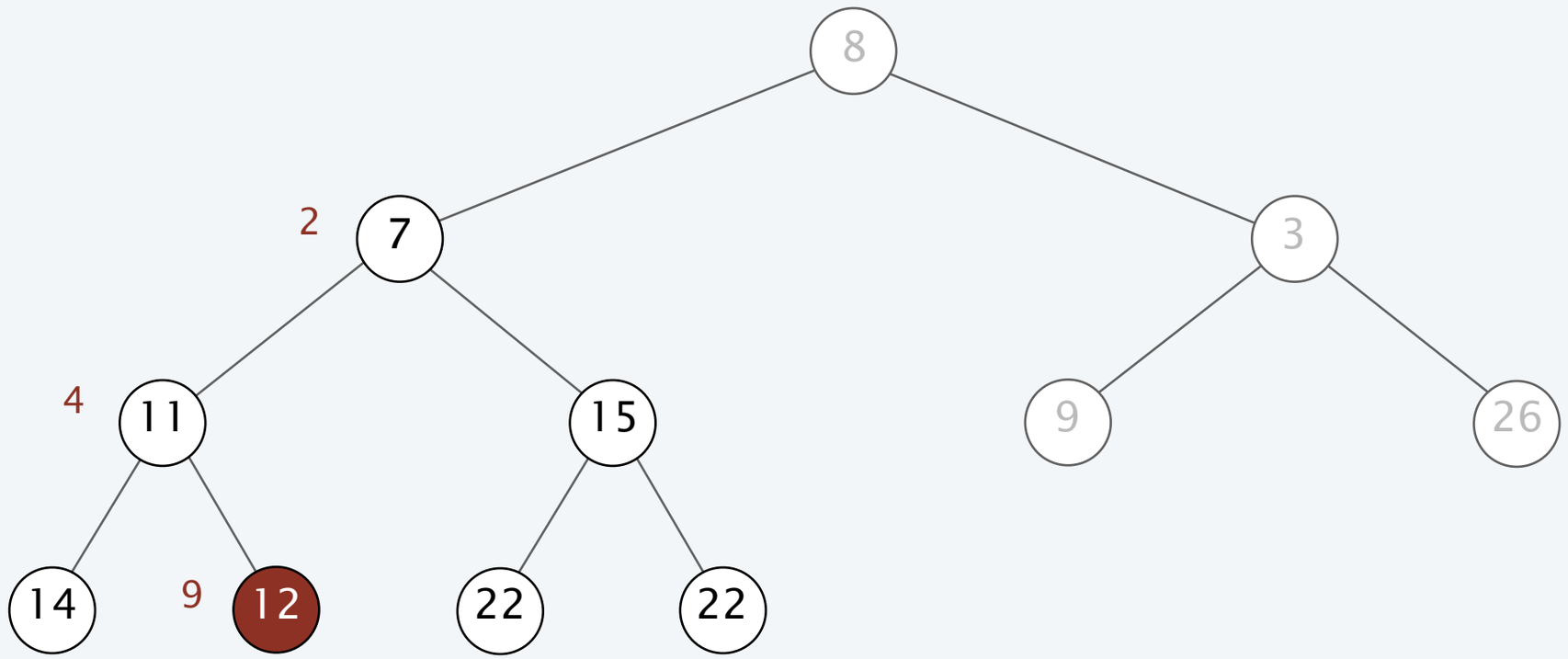


8	7	3	12	15	9	26	14	11	22	22
	2		4							

Heapify demo

Heapify. For each element in reverse-array order, sink it down.

sink 2

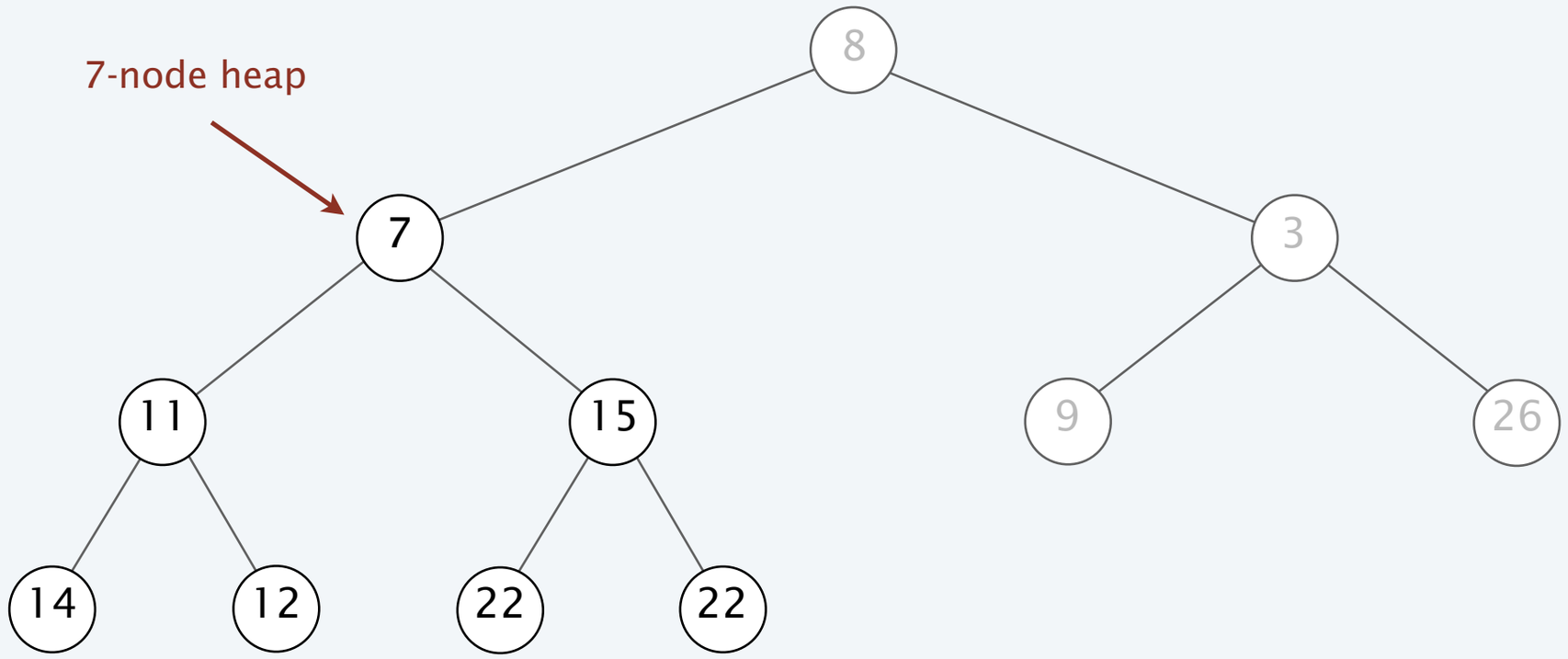


8	7	3	11	15	9	26	14	12	22	22
	2		4					9		

Heapify demo

Heapify. For each element in reverse-array order, sink it down.

sink 2

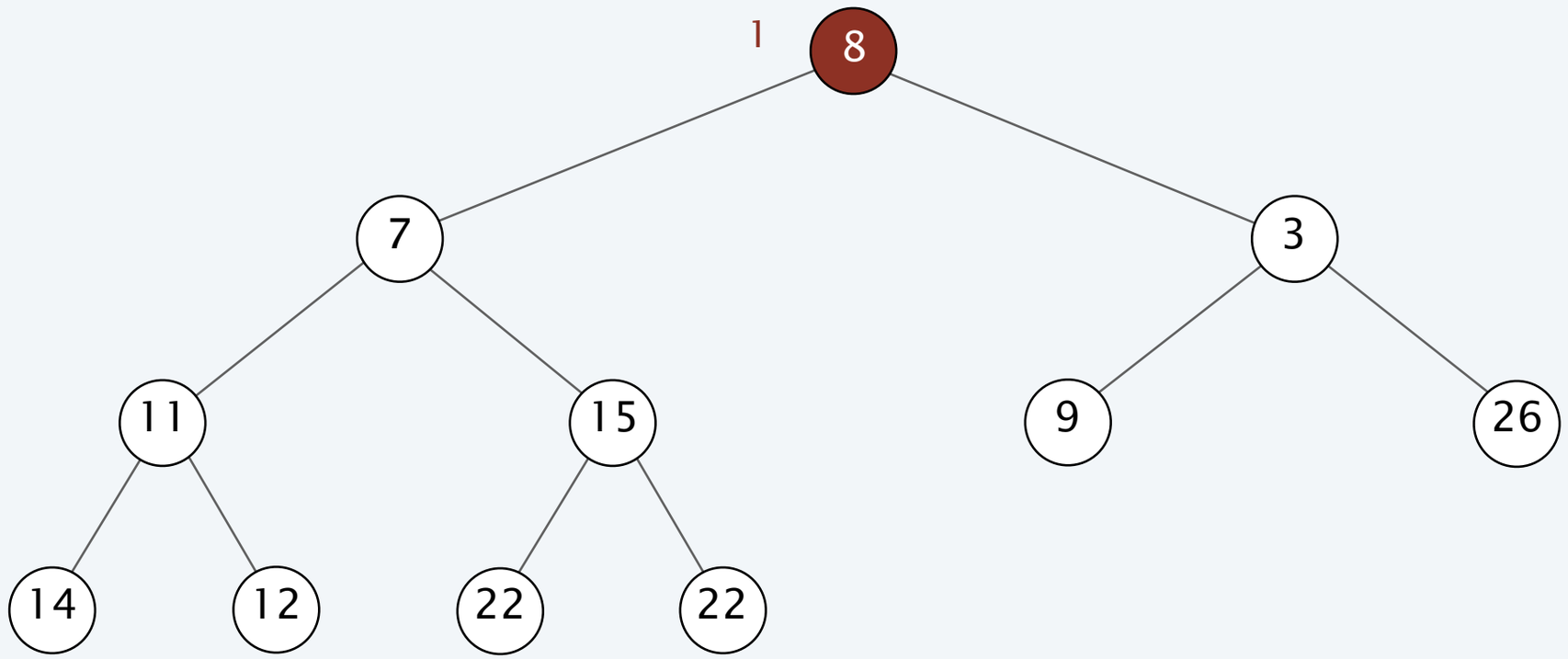


8	7	3	11	15	9	26	14	12	22	22

Heapify demo

Heapify. For each element in reverse-array order, sink it down.

sink 1

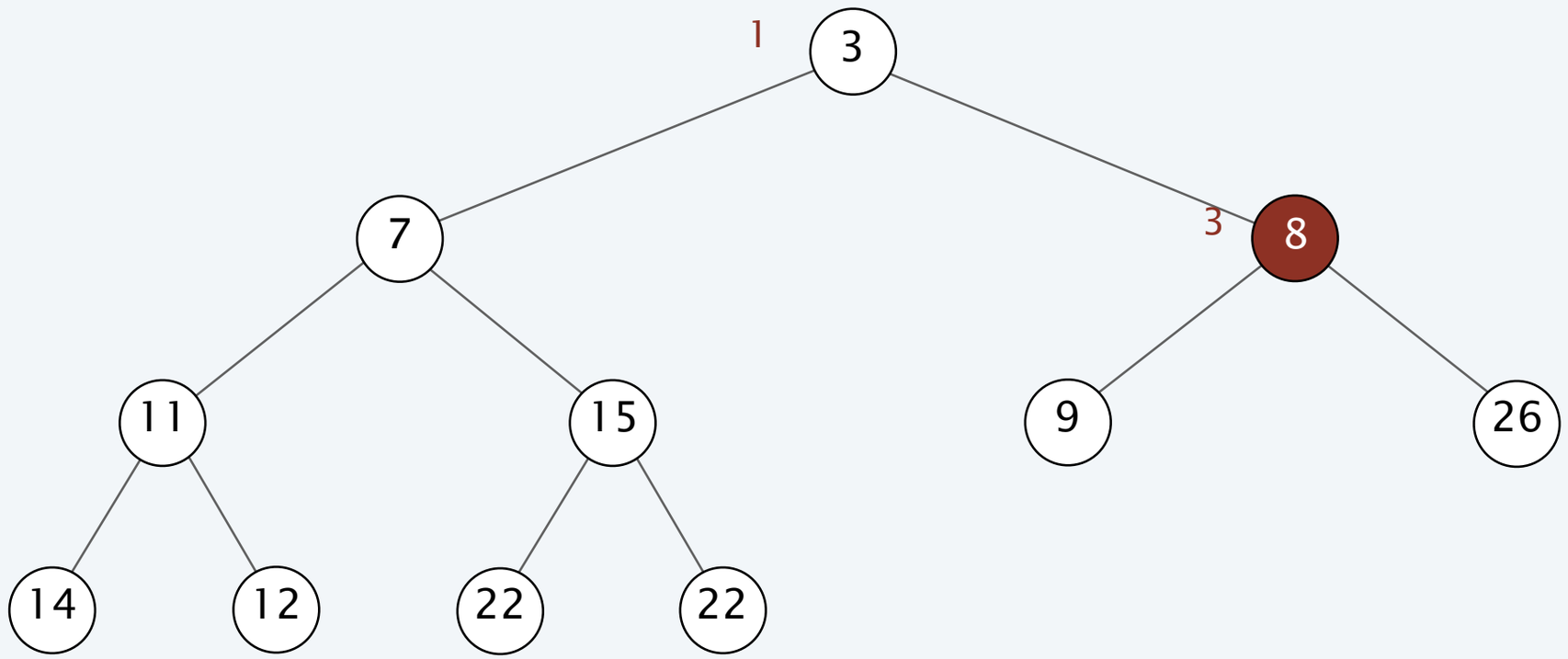


8	7	3	11	15	9	26	14	12	22	22
1										

Heapify demo

Heapify. For each element in reverse-array order, sink it down.

sink 1

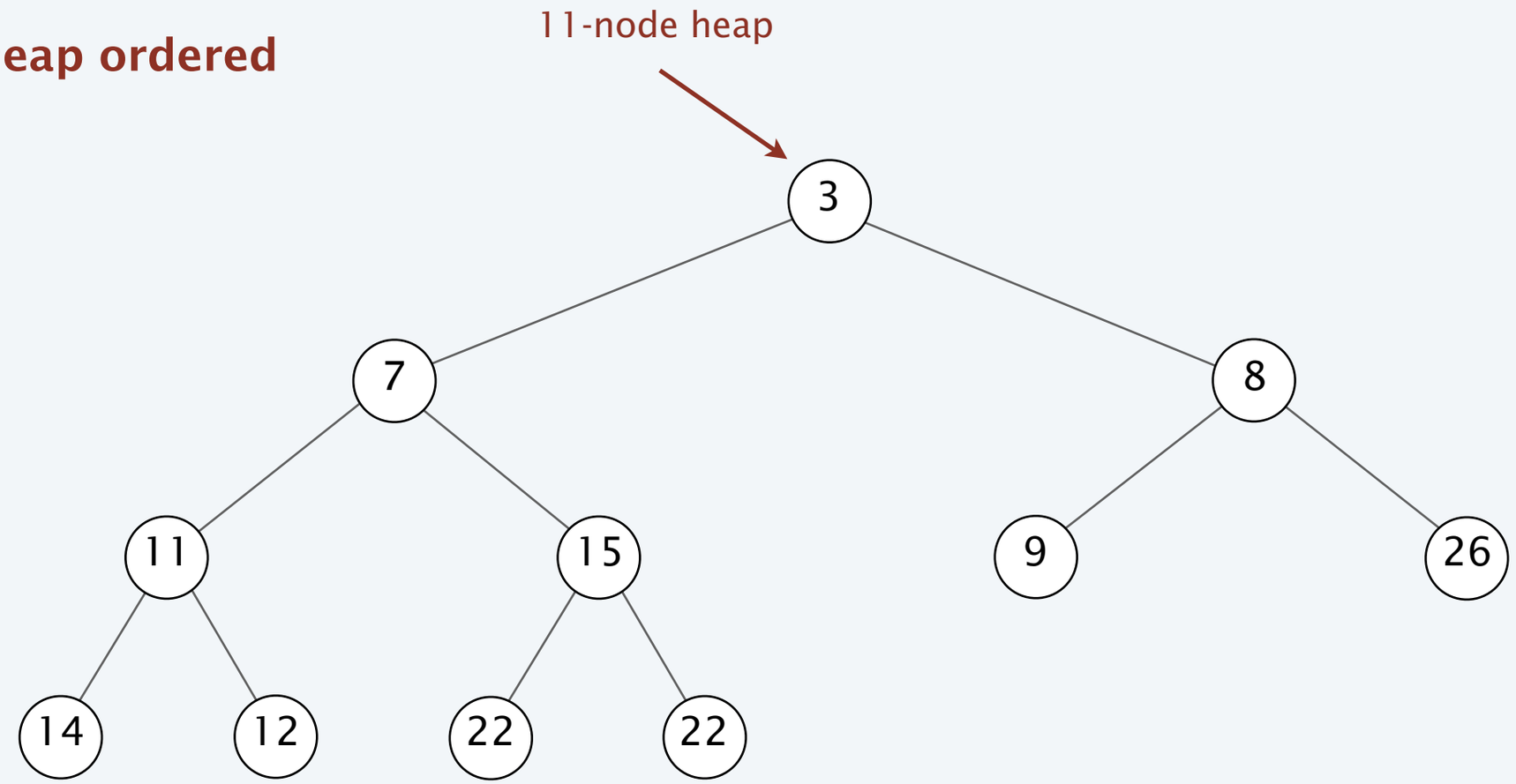


3	7	8	11	15	9	26	14	12	22	22
1		3								

Heapify demo

Heapify. For each element in reverse-array order, sink it down.

array is heap ordered



3	7	8	11	15	9	26	14	12	22	22

Priority queues performance cost summary

operation	linked list	binary heap
MAKE-HEAP	$O(1)$	$O(1)$
ISEMPTY	$O(1)$	$O(1)$
INSERT	$O(1)$	$O(\log n)$
EXTRACT-MIN	$O(n)$	$O(\log n)$
DECREASE-KEY	$O(1)$	$O(\log n)$
DELETE	$O(1)$	$O(\log n)$
MELD	$O(1)$	$O(n)$
FIND-MIN	$O(n)$	$O(1)$