Heapsort

Facts about heapsort:

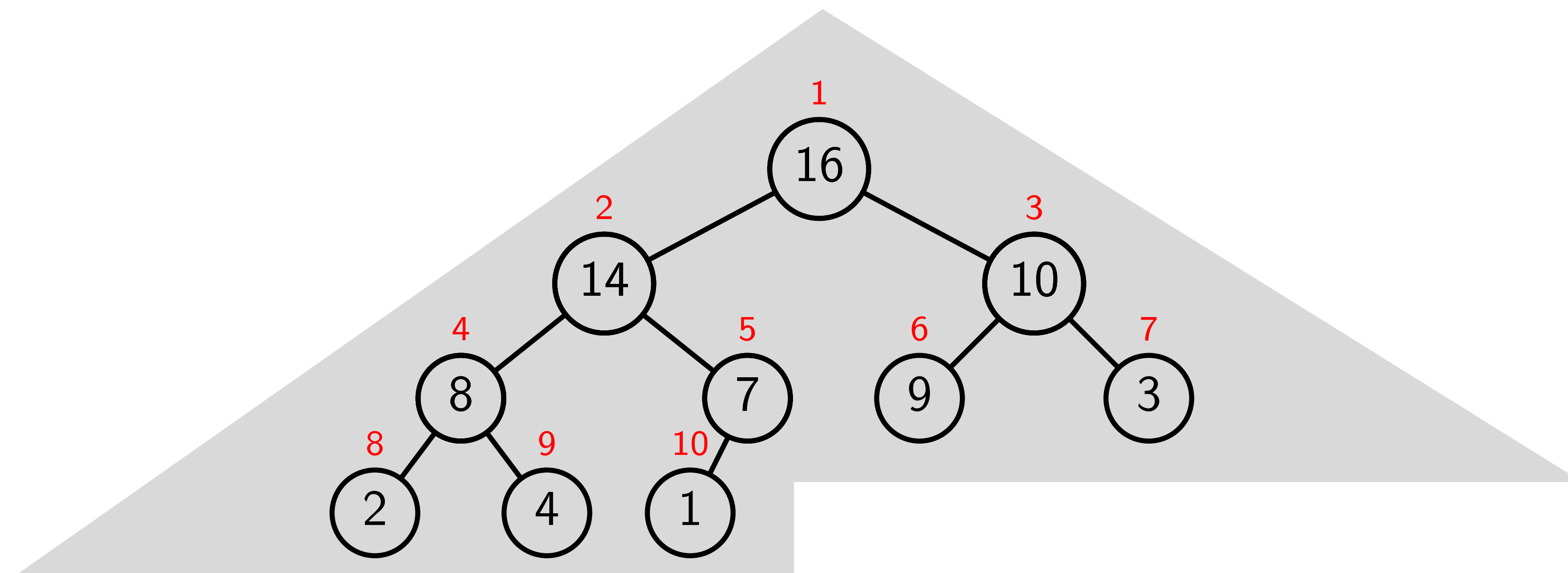
- $O(n \lg n)$ worst-case running time.
- Sorts in place: only $\Theta(1)$ extra memory needed.
- Thus heapsort combines the best of Merge sort and Quicksort.

The binary max-heap

Array representation:

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

Viewed as a binary tree:



The max-heap property

- For every node i (other than the root):

$$A[\textsc{Parent}(i)] \geq A[i]$$

- In other words, the biggest element is stored at the root.

- Finding parents and children is easy:

$$\begin{aligned}\textsc{Parent}(i) &= \lfloor i/2 \rfloor \\ \textsc{Left}(i) &= 2i \\ \textsc{Right}(i) &= 2i+1\end{aligned}$$

Maintaining the heap property

$\textsc{Max-Heapify}(A, i)$

$l \leftarrow \textsc{Left}(i)$

$r \leftarrow \textsc{Right}(i)$

if $l \leq \textit{heap-size}[A]$ and $A[l] > A[i]$

  then $\textit{largest} \leftarrow l$

  else $\textit{largest} \leftarrow i$

if $r \leq \textit{heap-size}[A]$ and $A[r] > A[\textit{largest}]$

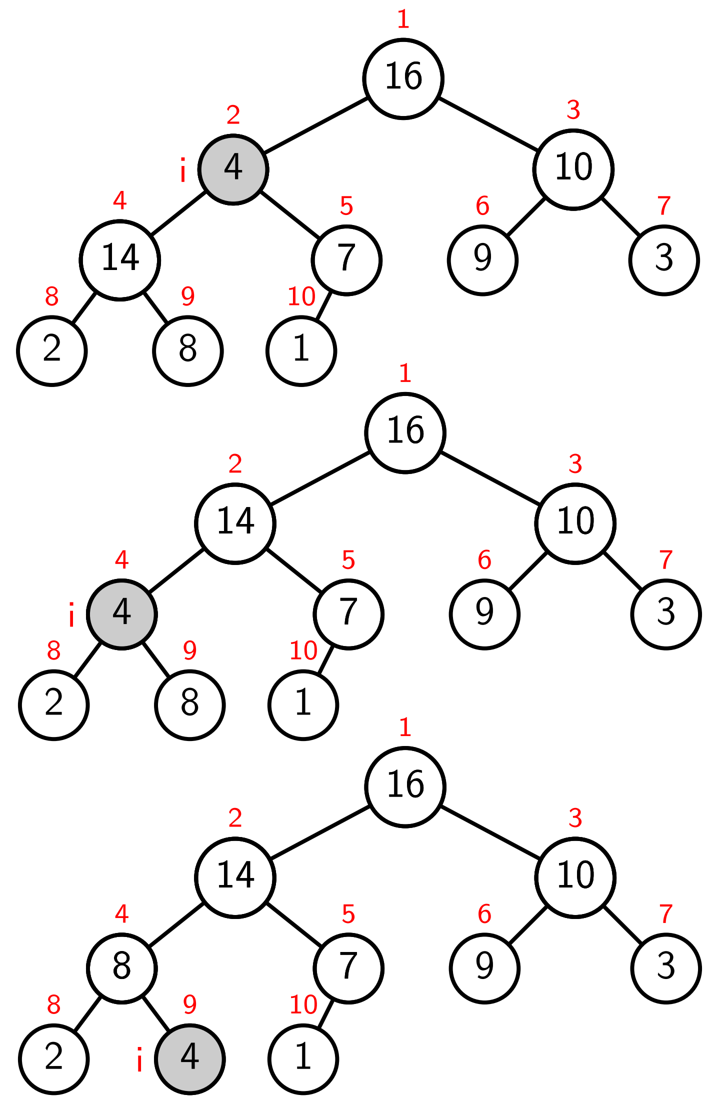
  then $\textit{largest} \leftarrow r$

$\triangleright\ A[\textit{largest}] = \max(A[i], A[l], A[r])$

if $\textit{largest} \neq i$

  then exchange $A[i] \leftrightarrow A[\textit{largest}]$

    $\textsc{Max-Heapify}(A, \textit{largest})$



Building a heap

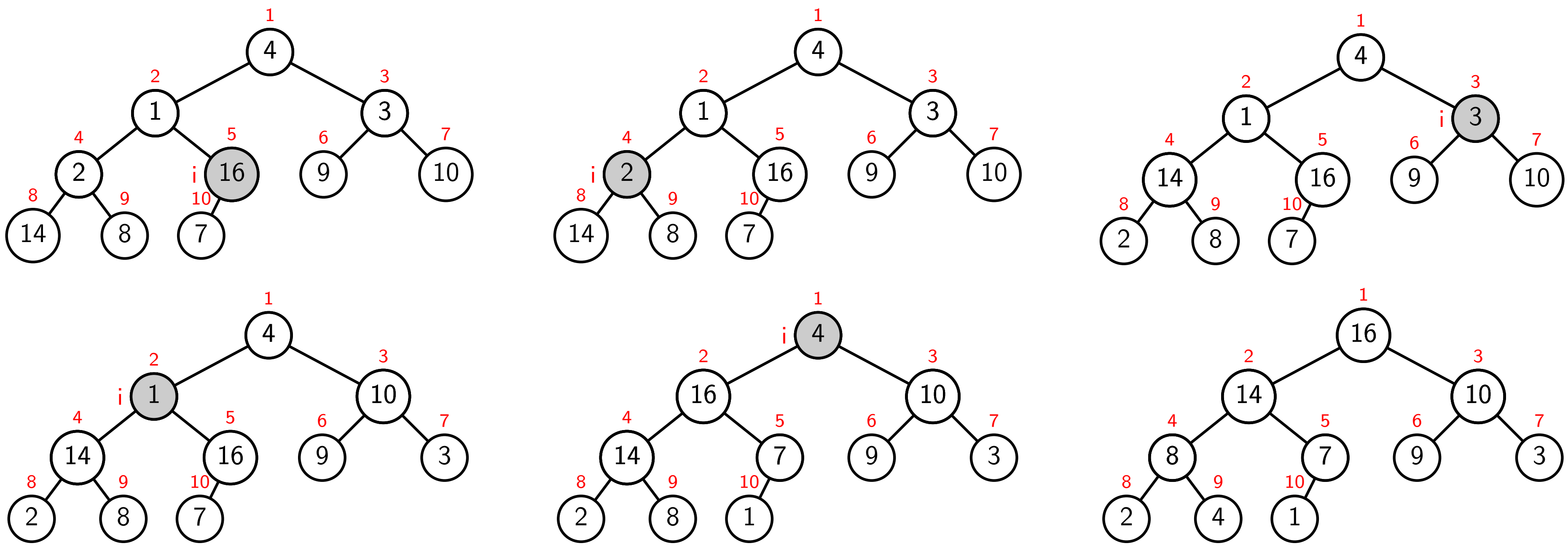
BUILD-MAX-HEAP(A)

$\quad$ *heap-size*$[A] \leftarrow$ *length*$[A]$

$\quad$ **for** $i \leftarrow \lfloor length[A]/2 \rfloor$ **downto** 1

$\quad\quad$ **do** MAX-HEAPIFY(A, i)

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



Analysis of Build-Max-Heap

- The max-heap is built bottom-up, using $\textsc{Max-Heapify}$.
- At the beginning, all $A[(\lfloor n/2 \rfloor + 1) \ldots n]$ are 1-element heaps.
- Every time around the **for**-loop, each node $i+1$, $i+2$, . . . , n is the root of a max-heap.
- The running time of $\textsc{Build-Max-Heap}$ is linear!

$$\sum_{h=0}^{\lfloor \lg n \rfloor} \lceil \frac{n}{2^{h+1}} \rceil \; O(h) = O(n \sum_{h=0}^{\lfloor \lg n \rfloor} \frac{n}{2^{h+1}}) = O(n \sum_{h=0}^{\infty} \frac{h}{2^h}) = O(n)$$

The heapsort algorithm

HEAPSORT(A)
1. BUILD-MAX-HEAP(A)
2. **for** $i \leftarrow length[A]$ **downto** 2
3. **do** exchange $A[1] \leftrightarrow A[i]$
4. $heap\text{-}size[A] \leftarrow heap\text{-}size[A] - 1$
5. MAX-HEAPIFY($A, 1$)

- After line 1, the maximum element is stored in $A[1]$.
- Exchange it with $A[n]$, then "disconnect" $A[n]$ from the heap.
- Restore the heap-property in $A[1 \ldots (n-1)]$ with MAX-HEAPIFY.
- Now the second largest element is in $A[1]$ again, etc.

Priority queues

- A *priority queue* maintains a set S of elements and supports the following operations:
  - $\textsc{Insert}(S, x)$ implements $S \leftarrow S \cup \{x\}$.
  - $\textsc{Maximum}(S)$ returns the largest element in S.
  - $\textsc{Extract-Max}(S)$ returns & removes the largest element in S.
  - $\textsc{Increase-Key}(S, x, k)$ increases the value of x to k.
- A priority queue can store, e.g., tasks with different priorities.
- Heaps can be used to represent priority queues!

Priority queues — implementation

$\text{Heap-Maximum}(A)$ $\qquad \triangleright\ O(1)$

1. **return** $A[1]$

$\text{Heap-Extract-Max}(A)$ $\qquad \triangleright\ O(\lg n)$

1. **if** $\mathit{heap\text{-}size}[A] < 1$
2. **then error** "heap underflow"
3. $\mathit{max} \leftarrow A[1]$
4. $A[1] \leftarrow A[\mathit{heap\text{-}size}[A]]$
5. $\mathit{heap\text{-}size}[A] \leftarrow \mathit{heap\text{-}size}[A] - 1$
6. $\text{Max-Heapify}(A, 1)$
7. **return** max

Priority queues — implementation

$\textsc{Heap-Increase-Key}(A, i, key)$ $\qquad \triangleright\ O(\lg n)$

1. **if** $key < A[i]$
2. **then error** "new key is smaller than current key"
3. $A[i] \leftarrow key$
4. **while** $i > 1$ and $A[\textsc{Parent}(i)] < A[i]$
5. **do** exchange $A[i] \leftrightarrow A[\textsc{Parent}(i)]$
6. $i \leftarrow \textsc{Parent}(i)$

$\textsc{Max-Heap-Insert}(A, key)$ $\qquad \triangleright\ O(\lg n)$

1. $\mathit{heap\text{-}size}[A] \leftarrow \mathit{heap\text{-}size}[A] + 1$
2. $A[\mathit{heap\text{-}size}[A]] \leftarrow -\infty$
3. $\textsc{Heap-Increase-Key}(A, \mathit{heap\text{-}size}[A], key)$