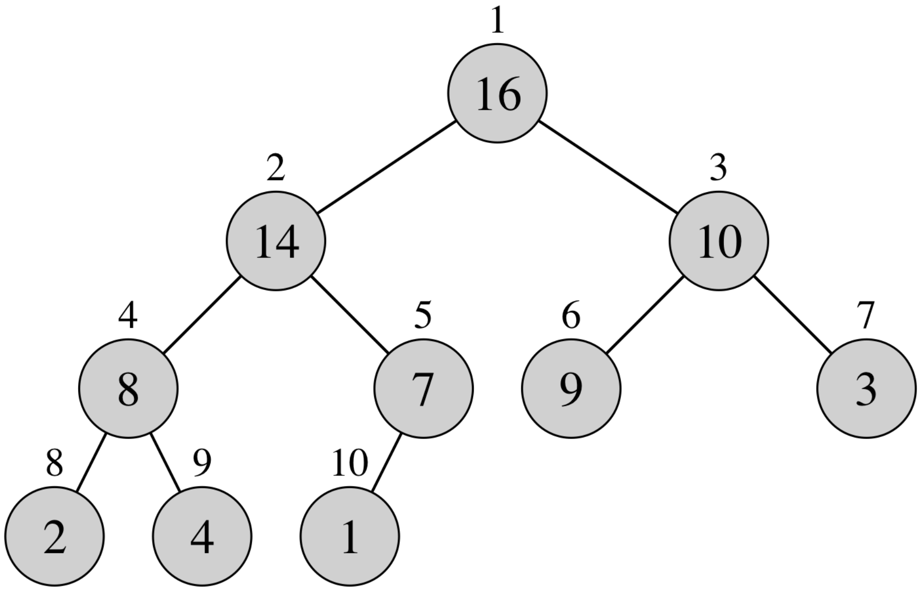
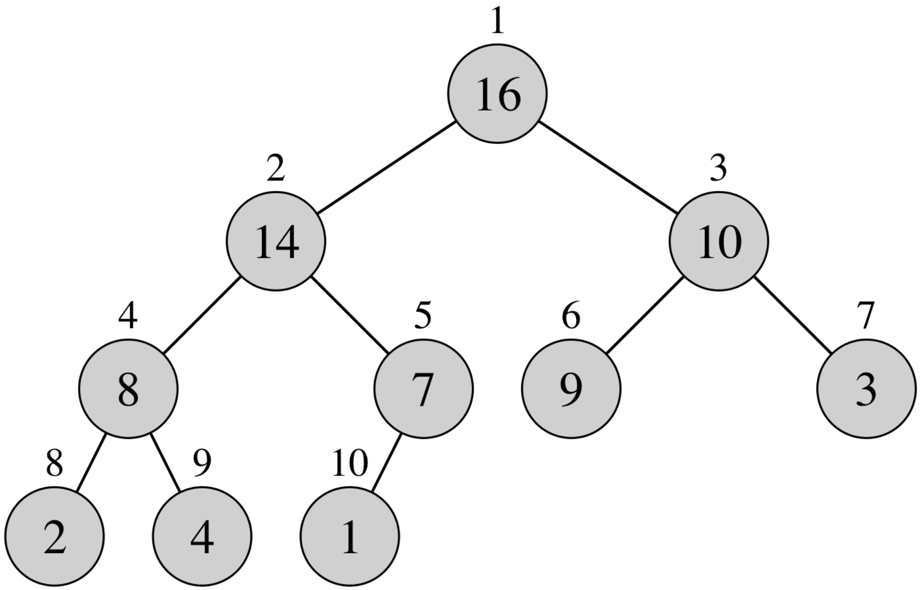
Heaps & heapsort

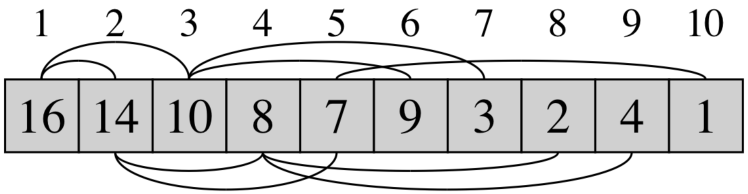
CLRS 6.1 – 6.5



(Max) Heaps

- A (binary) **heap** is an array object A that we can view as a nearly complete binary tree, where the **root is at $\boldsymbol{A}[\mathbf{1}]$**
- For any given node at index i, other related nodes can be found
  - *Parent*: at index $\left\lfloor \frac{i}{2} \right\rfloor$
  - *Left child*: at index $2i$
  - *Right child*: at index $2i+1$
- **Properties**:
  - **Nearly complete binary tree**: tree is completely filled on all levels except possibly the lowest, which is filled from the left up to a point
  - **Max-heap property**: for every node i other than the root, $A[\text{Parent}(i)] \geq A[i]$





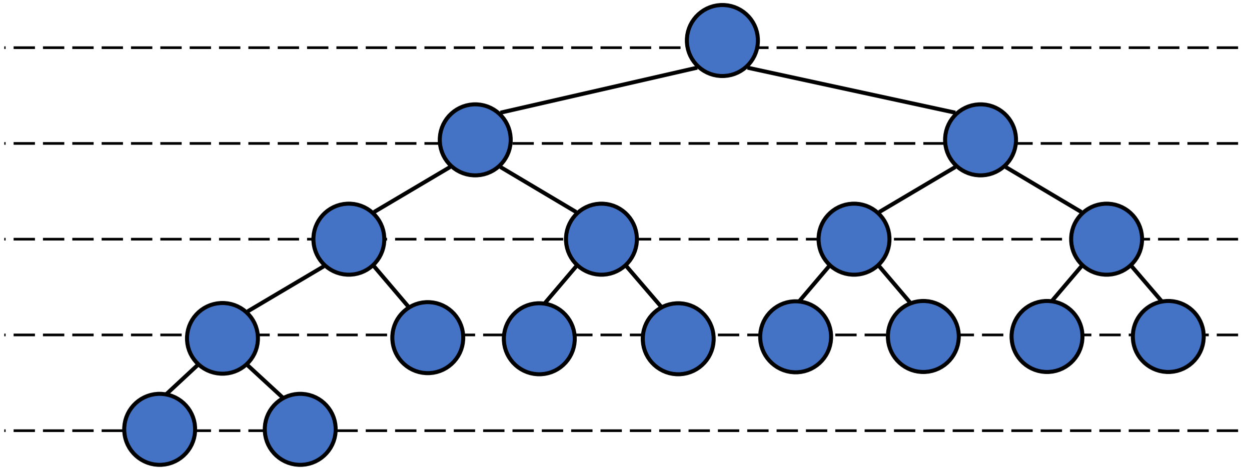
Heap height

Theorem: A heap containing $\boldsymbol{n}$ elements has height $O(\log n)$.

Proof:

- Let h be the height of a heap storing n keys.
- Since there are 2^i keys at depth $i = 0, \dots, h-1$ and at least one key at depth h, we have $n \geq 1 + 2 + 4 + \dots + 2^{h-1} + 1$.
- Thus, $n \geq 2^h$ and therefore $h \leq \log n$.

depth	keys
0	1
1	2
h–2	2^{h-2}
h–1	2^{h-1}
h	1



Heap operations

$O(\log n)$

- Max-Heap-Insert: *insert into heap*
- Heap-Extract-Max: *remove and return item with max key*
- Heap-Increase-Key: *increase value of particular key*
- **Max-Heapify**: *maintain max-heap property*

$O(1)$

- Heap-Maximum: *return (but do not remove) item with max key*

$O(n)$

- Build-Max-Heap: *construct a max-heap from an array of keys*

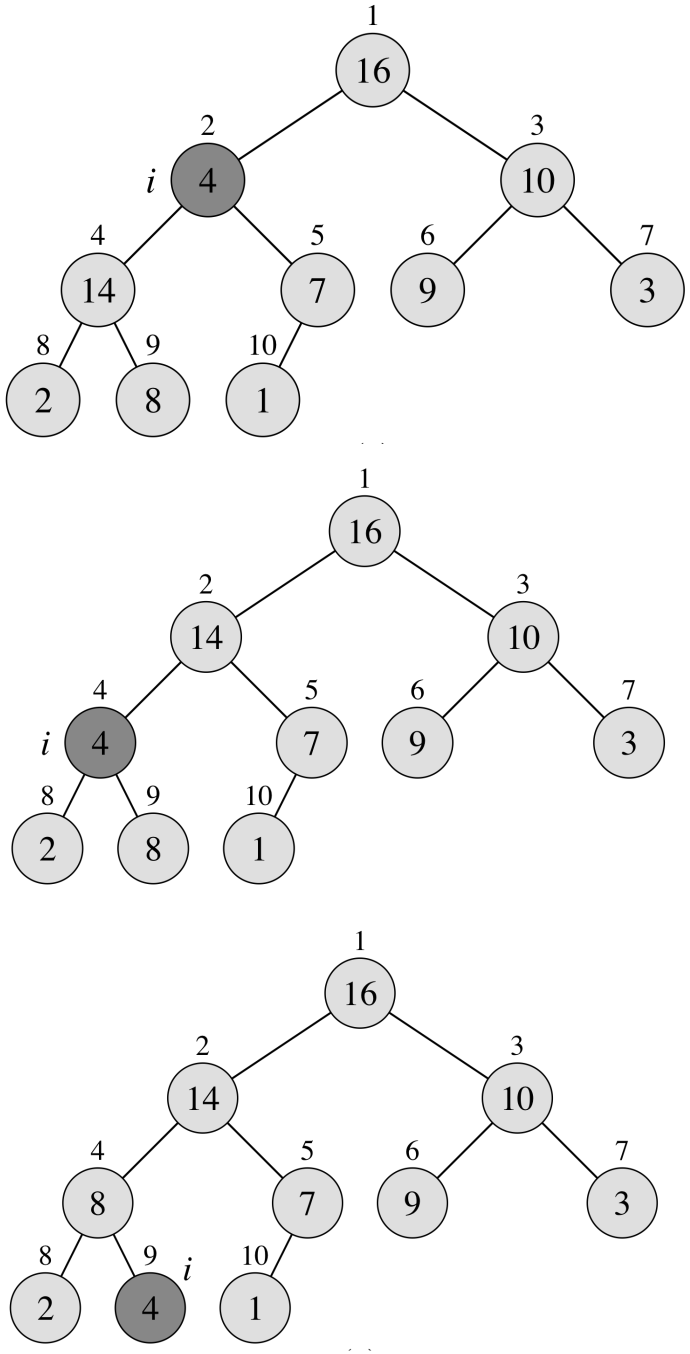
$O(n \log n)$

- Heapsort: *use a heap to sort an array of keys*

Max-Heapify

- Works on a particular node at index i.
- Assumption: Binary trees rooted at Left(i) and Right(i) are max-heaps, but possibly the node at index i might violate the max-heap property.
- **Idea**: While the max-heap property is violated, fix it by **floating down** the node

```
MAX-HEAPIFY(A, i, n)                    O(log n)
  $l$ = LEFT($i$)
  $r$ = RIGHT($i$)
  if $l \le n$ and $A[l] > A[i]$
        $largest = l$
  else $largest = i$
  if $r \le n$ and $A[r] > A[largest]$
        $largest = r$
  if $largest \neq i$
        exchange $A[i]$ with $A[largest]$
        MAX-HEAPIFY($A$, $largest$, $n$)
```



Inserting a single element

- Place it at the end of the array (next empty node of tree)
- While the max-heap property is violated, fix it by **floating up** the node.

```
MAX-HEAP-INSERT(A, key, n)        $O(\log n)$
  $n = n + 1$
  $A[n] = -\infty$
  HEAP-INCREASE-KEY(A, n, key)
```

```
HEAP-INCREASE-KEY(A, i, key)        $O(\log n)$
  if $key < A[i]$
        error "new key is smaller than current key"
  $A[i] = key$
  while $i > 1$ and $A[\text{PARENT}(i)] < A[i]$
        exchange $A[i]$ with $A[\text{PARENT}(i)]$
        $i = \text{PARENT}(i)$
```

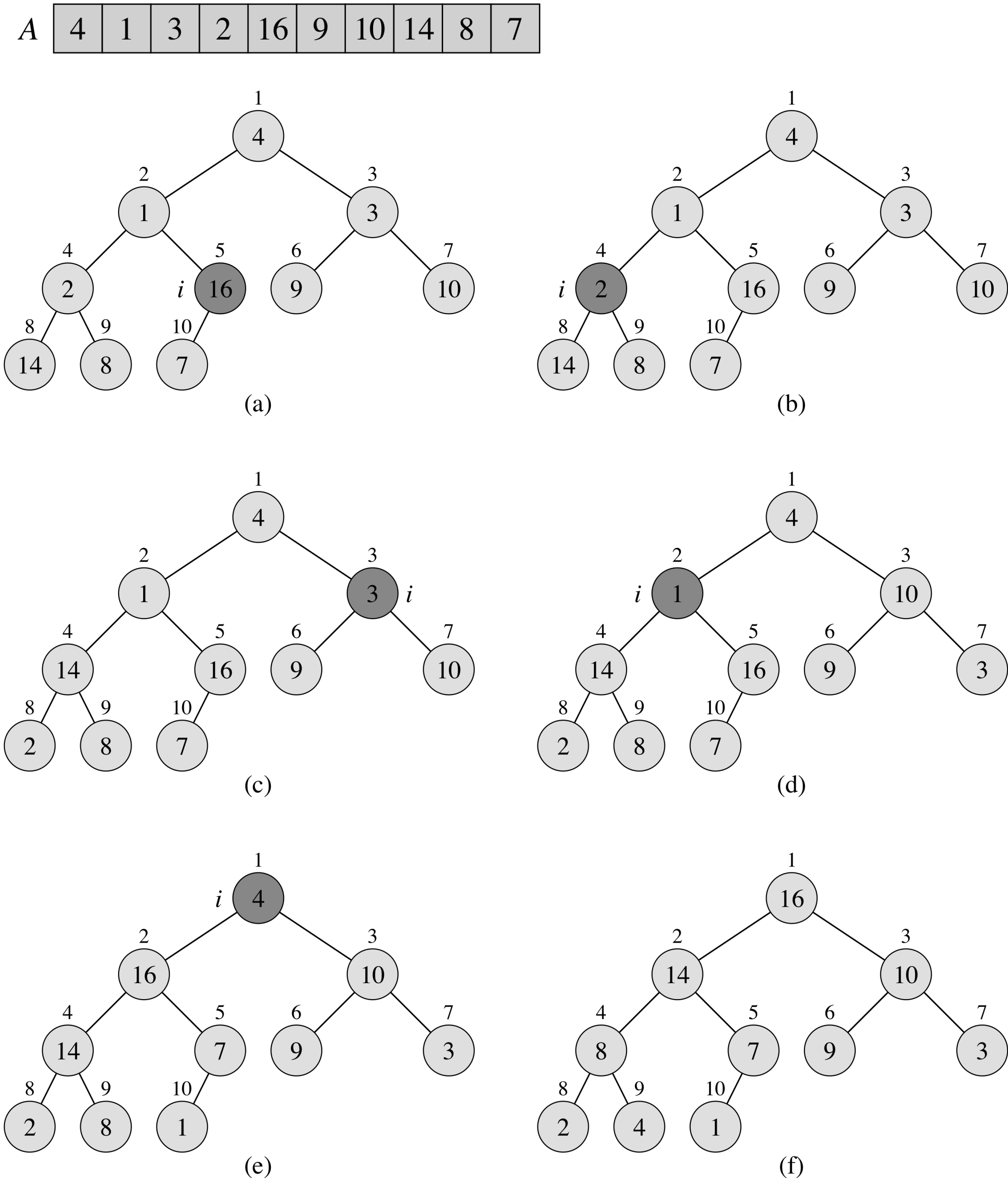
Visualization ("insert"): http://btv.melezinek.cz/binary-heap.html

Constructing a heap from an array of elements

- Take advantage of the fact that all elements are known in advance
- Repeatedly use max-heapify

$O(n)$

```
BUILD-MAX-HEAP$(A, n)$
  for $i = \lfloor n/2 \rfloor$ downto 1
      MAX-HEAPIFY$(A, i, n)$
```



A	4	1	3	2	16	9	10	14	8	7

Visualization (“build heap”): http://btv.melezinek.cz/binary-heap.html

Extracting the maximum

- Remove the maximum (known to be the root node at A[1])
- Exchange it with the last item
- Fix the max-heap property

```
HEAP-EXTRACT-MAX(A, n)        $O(\log n)$
    if $n < 1$
        error "heap underflow"
    $max = A[1]$
    $A[1] = A[n]$
    $n = n - 1$
    MAX-HEAPIFY(A, 1, n)          // remakes heap
    return max
```

Visualization ("extract max"): http://btv.melezinek.cz/binary-heap.html

Heapsort

- Efficiently build a heap
- Repeatedly remove the maximum item, placing it at the end of the array
- Repeat process with remaining part of the heap

$O(n \log n)$

```
HEAPSORT$(A, n)$
  BUILD-MAX-HEAP$(A, n)$
  for $i = n$ downto 2
      exchange $A[1]$ with $A[i]$
      MAX-HEAPIFY$(A, 1, i - 1)$
```

Visualization ("heap sort"): http://btv.melezinek.cz/binary-heap.html

Application to Priority Queues

- A **priority queue** stores a collection of (key, element) pairs and supports
  - Insert
  - Maximum (Minimum)
  - Extract-Max (Extract-Min)

- **Easy to sort using a priority queue as auxiliary data structure**
  - Insert all items into priority queue
  - One-by-one, call extract-max and place item at the beginning of list

- This generic priority-queue approach to sorting encapsulates common sorting algorithms, depending on the **implementation** of the priority queue
  - Use heap → heapsort
  - Use unsorted list → selection sort
  - Use sorted list → insertion sort