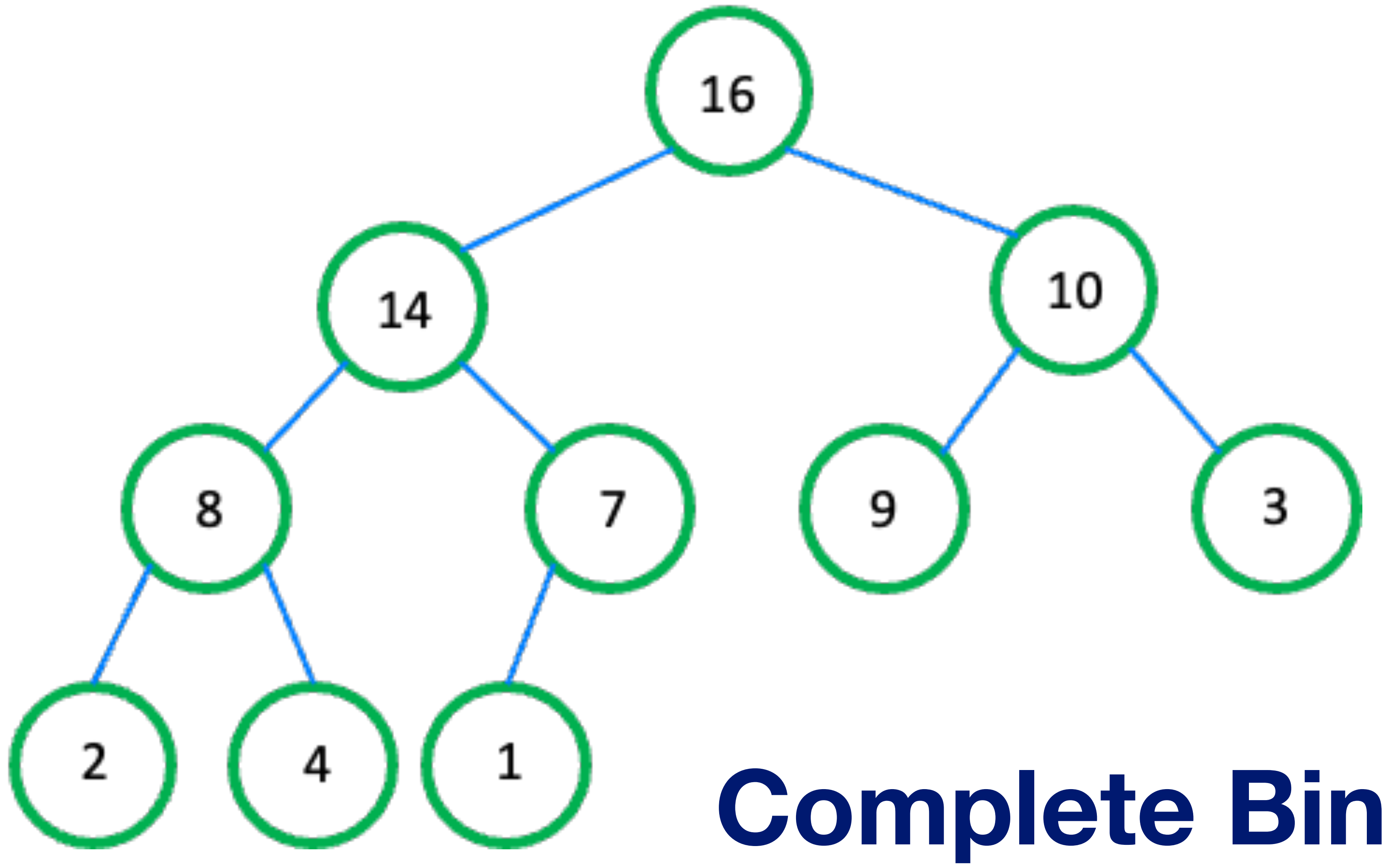
Programming Practice for Data Science

Lecture 8: Heap and Priority Queue (11/22/24)

Taesup Kim
Graduate School of Data Science
Seoul National University

Heap
Definition

- **The (binary) heap data structure is an array object**
  - view as a nearly **complete binary tree** (filled from left up to a point)
  - Each node of the tree corresponds to an element of the array



Complete Binary Tree

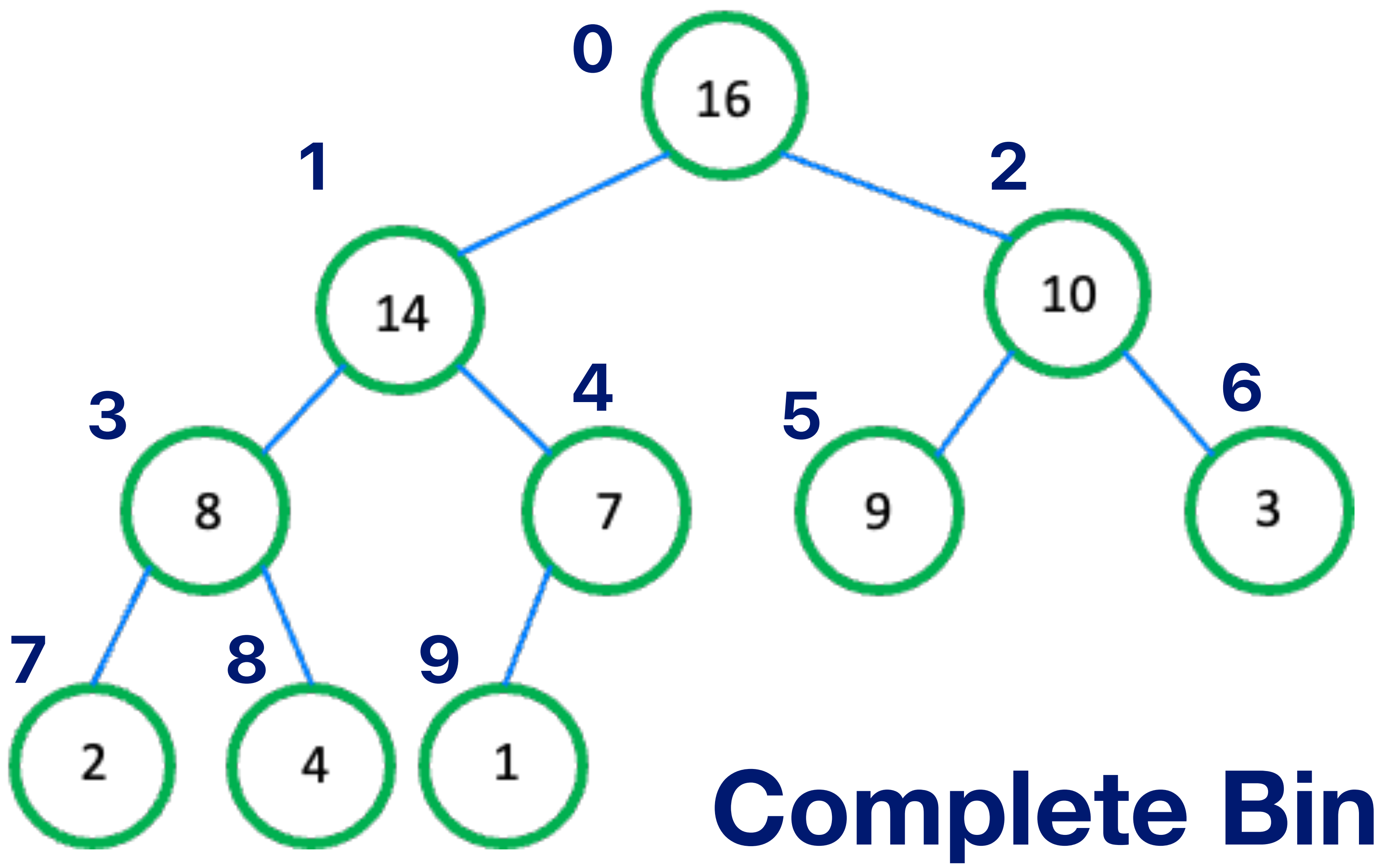
Array

0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**
16	14	10	8	7	9	3	2	4	1

Heap

Definition

- **The (binary) heap data structure is an array object**
  - The root of the tree is A[0]
  - **parent**(i): floor((i - 1) / 2), **left-child**(i): 2 * i + 1, **right-child**(i): 2 * i + 2



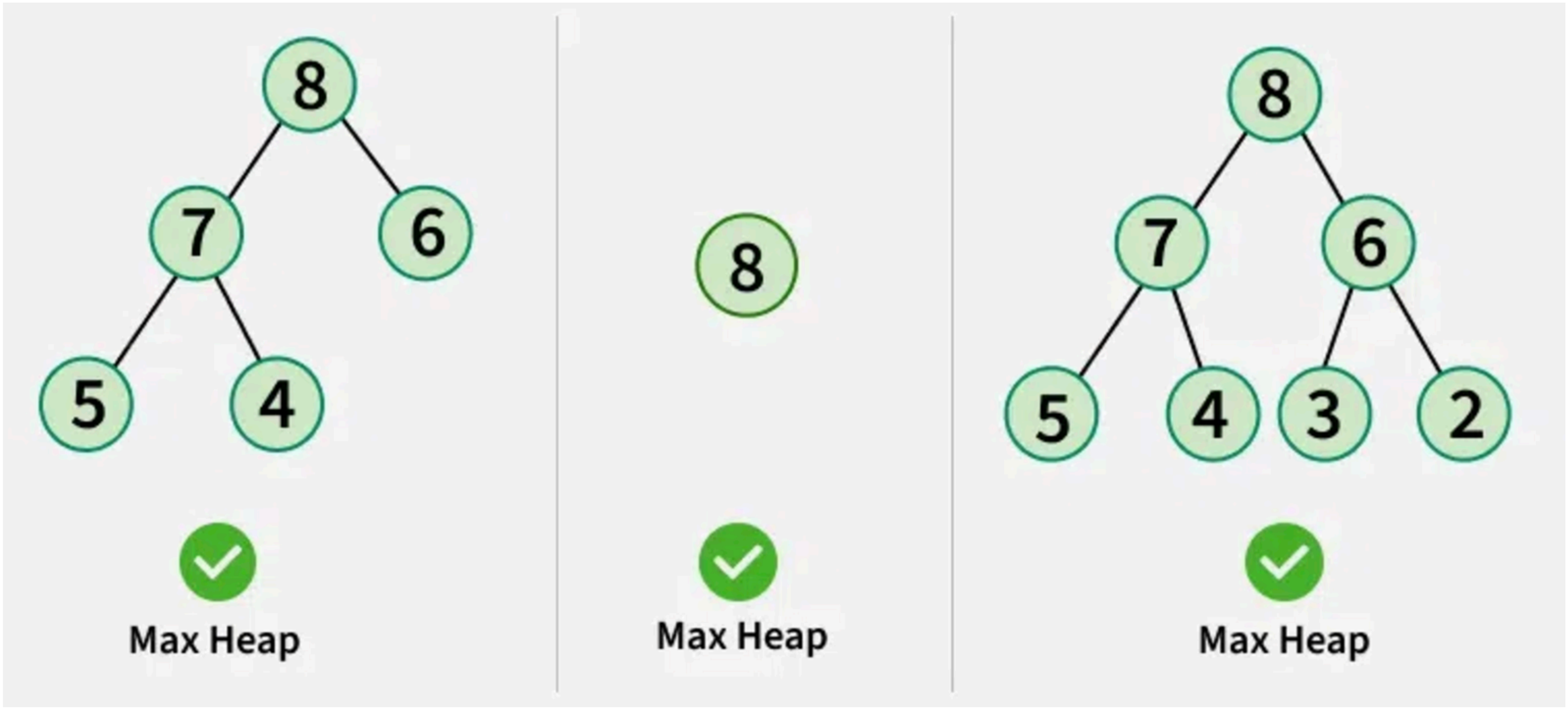
Complete Binary Tree

Array

0	1	2	3	4	5	6	7	8	9
16	14	10	8	7	9	3	2	4	1

Heap
Definition

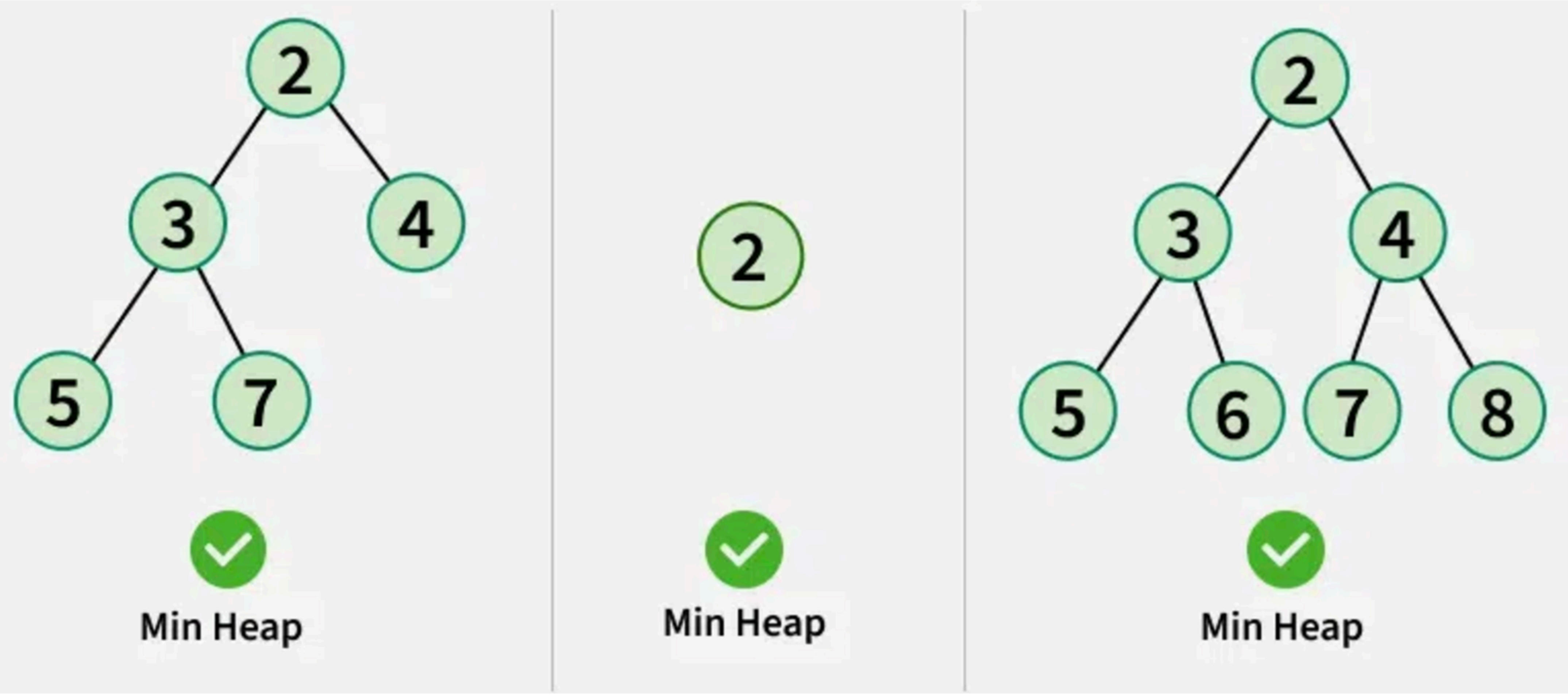
- **Max-Heap**
  - For every node i other than the root, A[parent(i)] ≥ A[I]
  - The largest element at the root A[0]



Heap

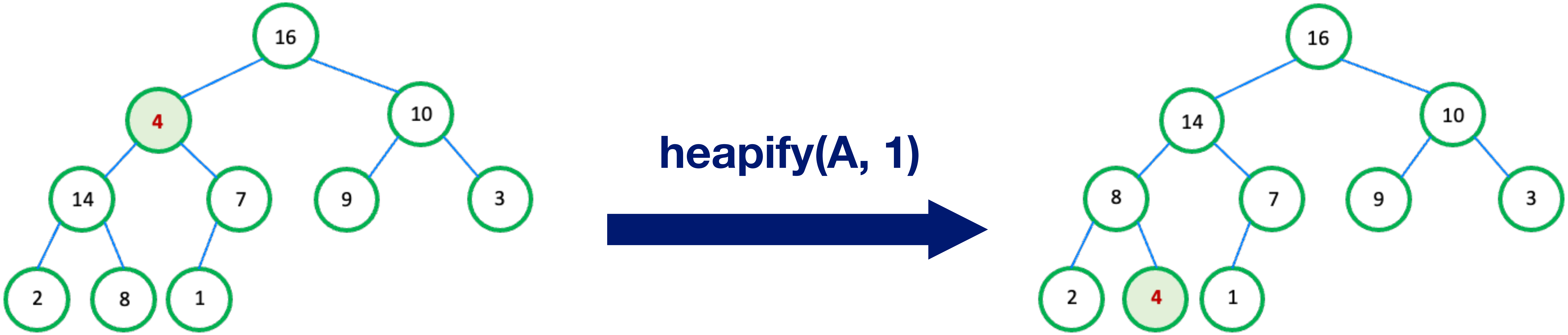
Definition

- **Min-Heap**
  - For every node i other than the root, A[parent(i)] ≤ A[I]
  - The smallest element at the root A[0]



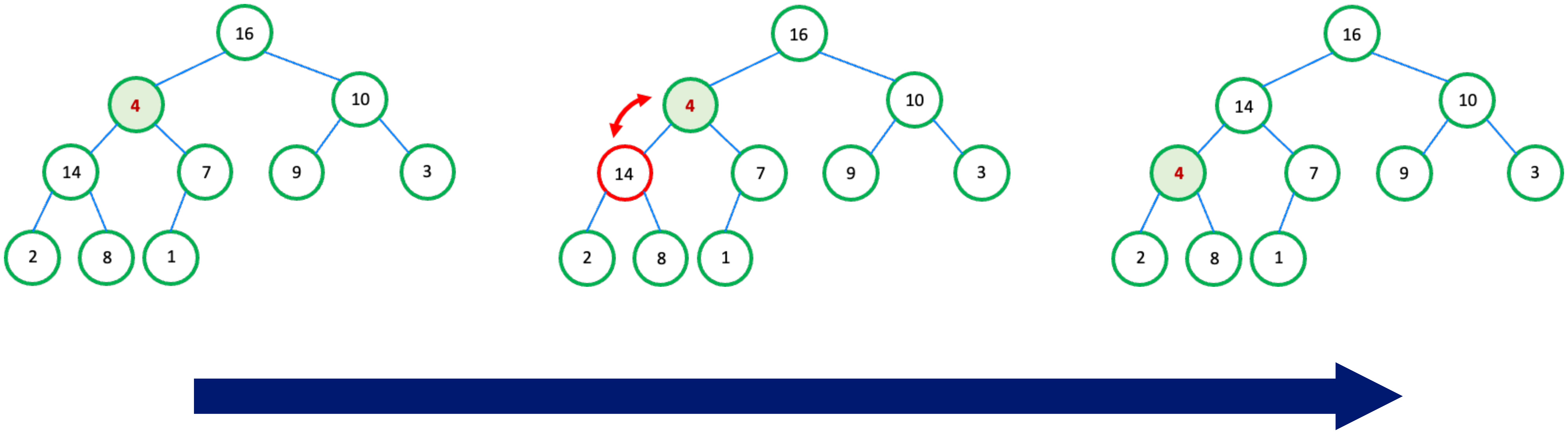
Heap
Definition

- **Max Heapify**
  - the value at A[i] “**float down**” in the max-heap,
    so that the subtree rooted at index i obeys the **max-heap property**



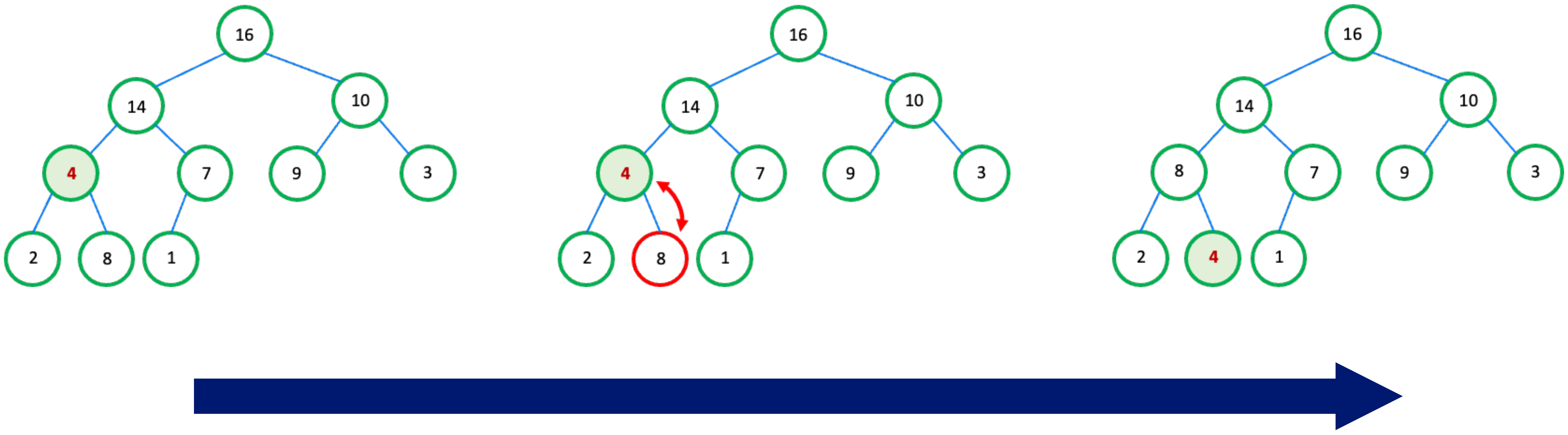
Heap
Definition

- **Max Heapify**
  - the value at A[i] "**float down**" in the max-heap, so that the subtree rooted at index i obeys the **max-heap property**



Heap
Definition

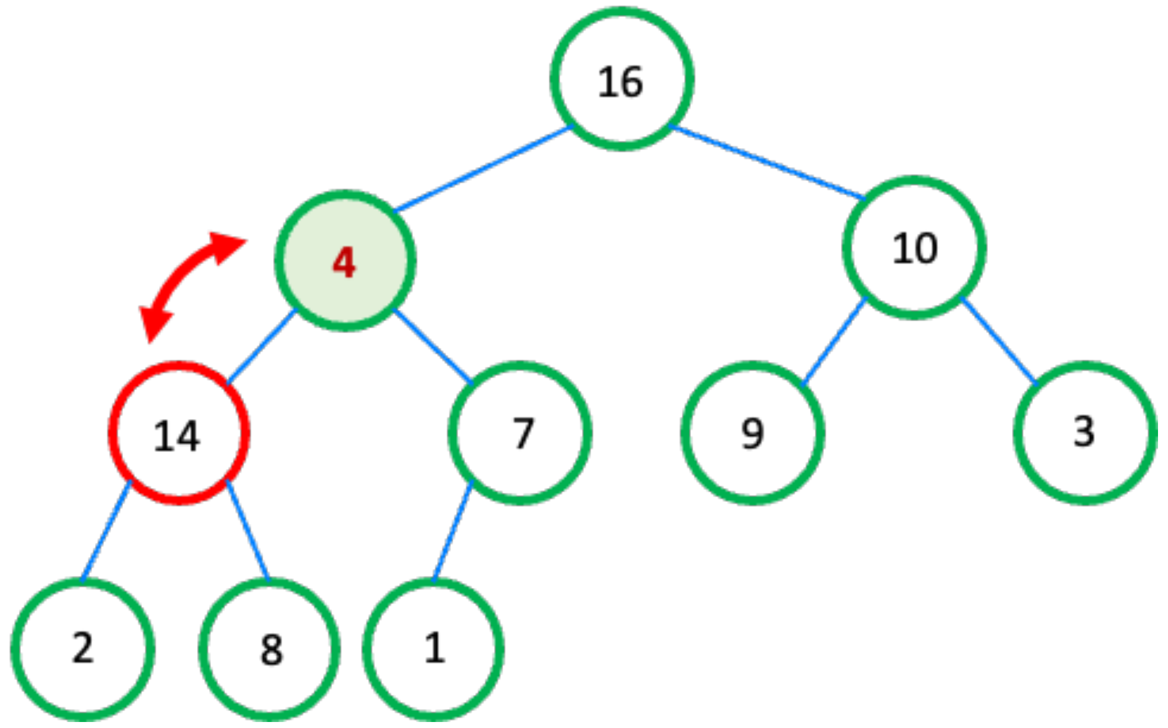
- **Max Heapify**
  - the value at A[i] "**float down**" in the max-heap, so that the subtree rooted at index i obeys the **max-heap property**



Heap

Definition

- **Max Heapify**
  - the value at A[i] "**float down**" in the max-heap,
    so that the subtree rooted at index i obeys the **max-heap property**

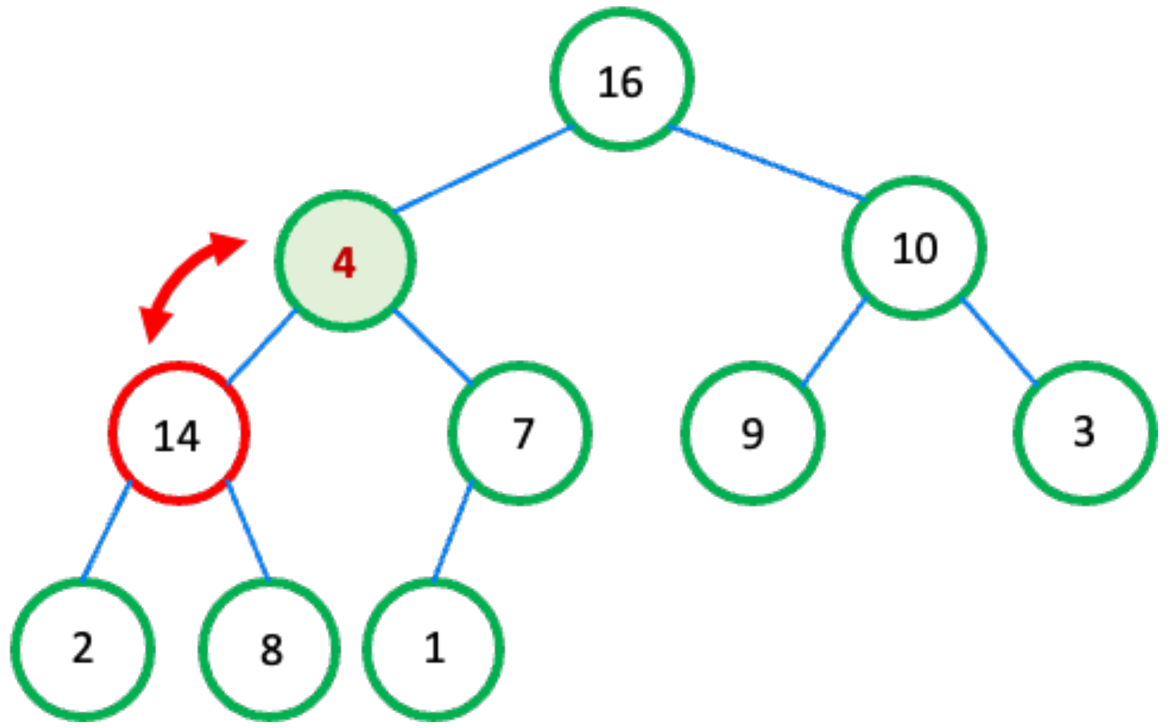



```
MAX-HEAPIFY(A, i)
l = LEFT(i)
r = RIGHT(i)
if l ≤ A.heap-size and A[l] > A[i]
    largest = l
else largest = i
if r ≤ A.heap-size and A[r] > A[largest]
    largest = r
if largest ≠ i
    exchange A[i] with A[largest]
    MAX-HEAPIFY(A, largest)
```

Heap

Definition

- **Max Heapify**
  - the value at A[i] "**float down**" in the max-heap,
    so that the subtree rooted at index i obeys the **max-heap property**

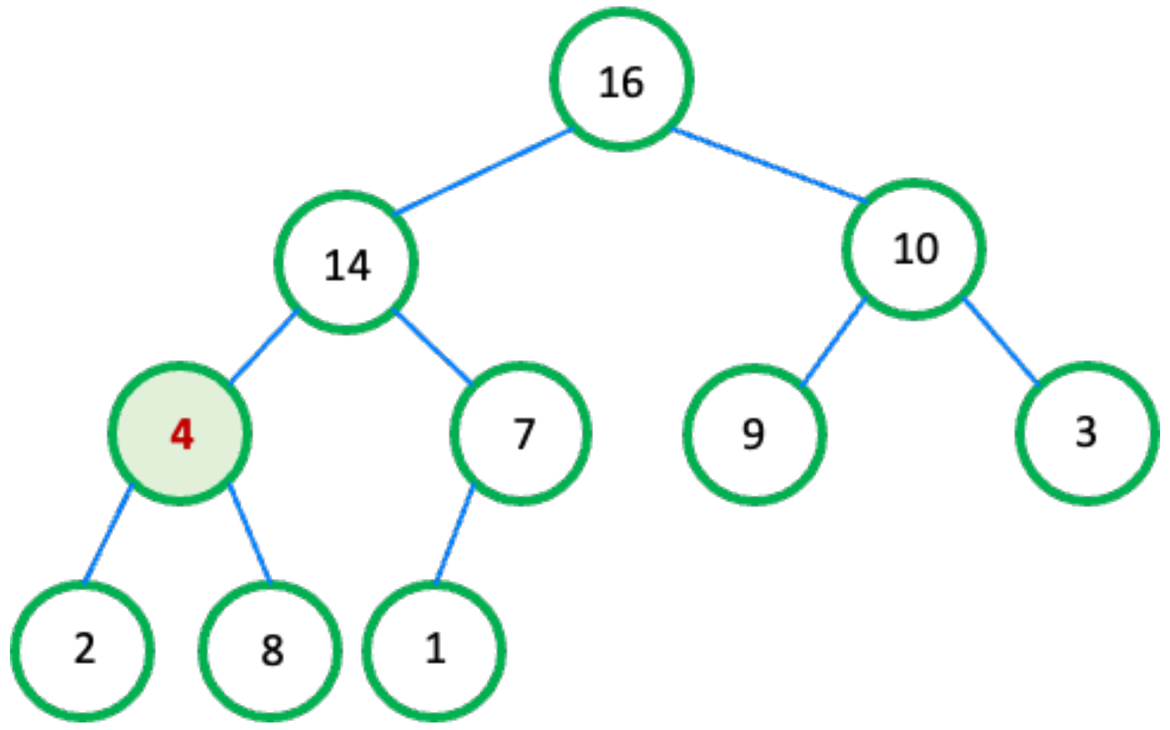



```
MAX-HEAPIFY(A, i)
l = LEFT(i)
r = RIGHT(i)
if l ≤ A.heap-size and A[l] > A[i]
    largest = l
else largest = i
if r ≤ A.heap-size and A[r] > A[largest]
    largest = r
if largest ≠ i
    exchange A[i] with A[largest]
    MAX-HEAPIFY(A, largest)
```

Heap

Definition

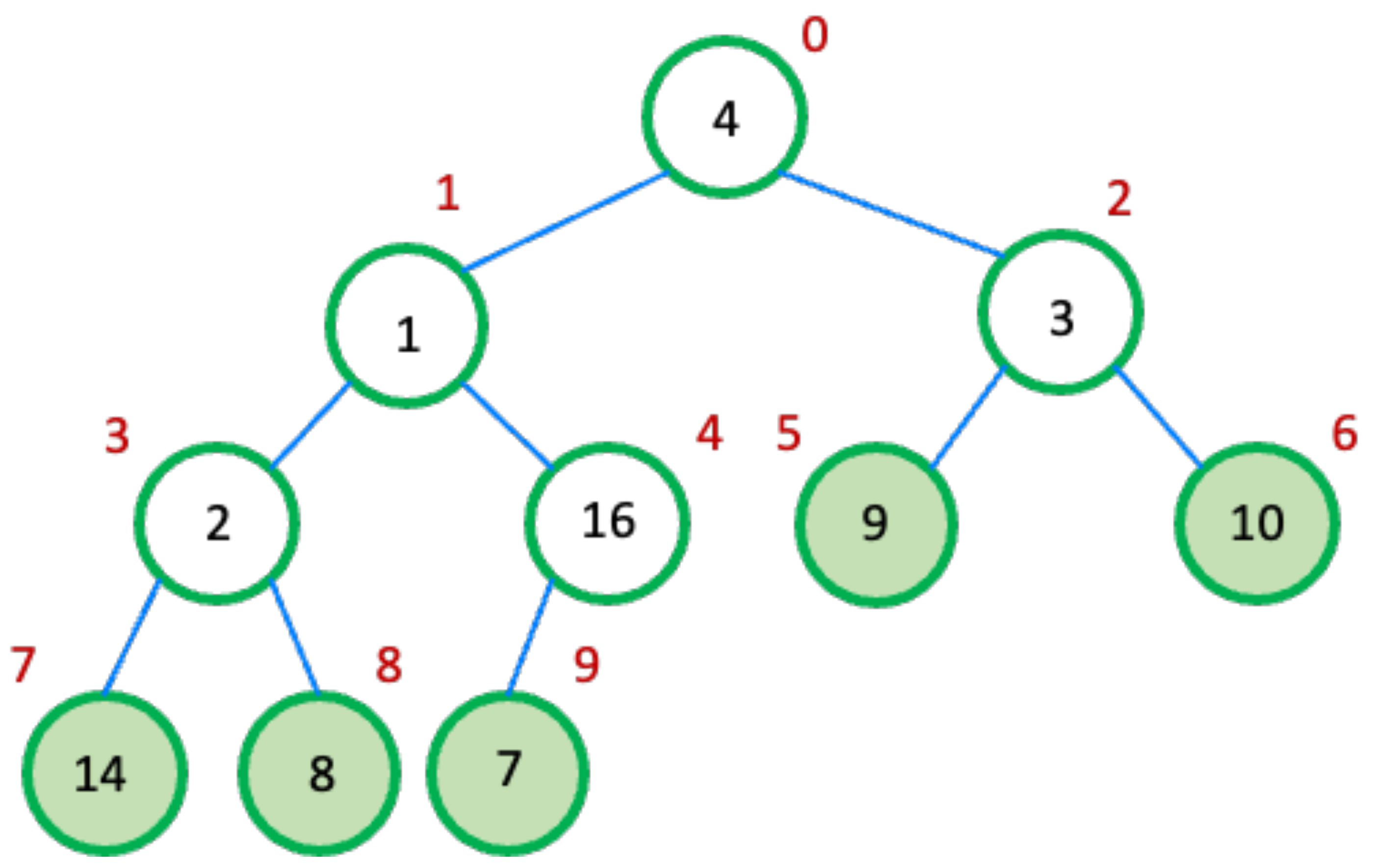
- **Max Heapify**
  - the value at A[i] “**float down**” in the max-heap,
    so that the subtree rooted at index i obeys the **max-heap property**




```
MAX-HEAPIFY(A, i)
l = LEFT(i)
r = RIGHT(i)
if l ≤ A.heap-size and A[l] > A[i]
    largest = l
else largest = i
if r ≤ A.heap-size and A[r] > A[largest]
    largest = r
if largest ≠ i
    exchange A[i] with A[largest]
    MAX-HEAPIFY(A, largest)
```

Heap
Definition

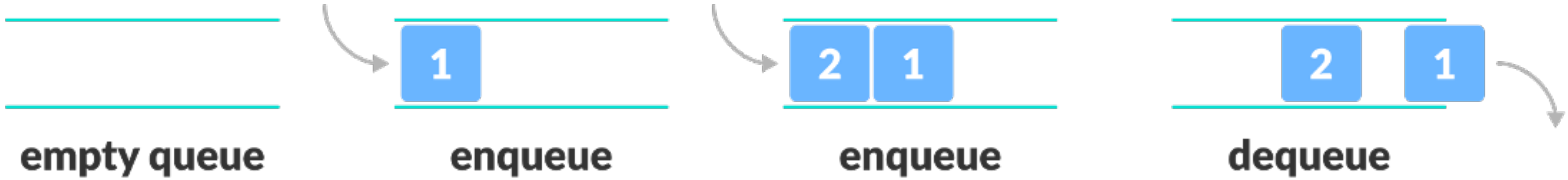
- **Building Max-Heap**
  - by calling max_heapify in a bottom-up manner




```
BUILD-MAX-HEAP(A, n)
    A.heap-size = n
    for i = ⌊n/2⌋ downto 1
        MAX-HEAPIFY(A, i)
```

Priority Queue
Definition

- **Queue**
  - First In First Out (FIFO)



Priority Queue

Definition

- **Priority Queue with Heaps**
  - Each element with an associated value called a key (priority)
  - Queue order defined by **key values**, not FIFO
  - Max-heap: higher key value = higher priority

Priority Queue
Definition

- **Priority Queue with Heaps**
  - ENQUEUE: insert to queue
  - DEQUEUE: extract from queue
  - INCREASE-KEY: modify priority

Priority Queue

Definition

- **Priority Queue with Heaps**
  - ENQUEUE: insert to queue
  - DEQUEUE: extract from queue
  - **INCREASE-KEY**: modify priority

MAX-HEAP-INCREASE-KEY(A, x, k)

if $k < x.key$
    error "new key is smaller than current key"
$x.key = k$
find the index i in array A where object x occurs
while $i > 1$ and $A[\text{PARENT}(i)].key < A[i].key$
    exchange $A[i]$ with $A[\text{PARENT}(i)]$, updating the information that maps priority queue objects to array indices
    $i = \text{PARENT}(i)$

"Move up"
swap(A[i], A[parent(i)])

Priority Queue

Definition

- **Priority Queue with Heaps**
  - ENQUEUE: insert to queue
  - **DEQUEUE**: extract from queue
  - INCREASE-KEY: modify priority

```
MAX-HEAP-EXTRACT-MAX(A)
max = MAX-HEAP-MAXIMUM(A)
A[1] = A[A.heap-size]
A.heap-size = A.heap-size - 1
MAX-HEAPIFY(A, 1)
return max
```

Priority Queue

Definition

- **Priority Queue with Heaps**
  - **ENQUEUE**: insert to queue
  - DEQUEUE: extract from queue
  - INCREASE-KEY: modify priority

```
MAX-HEAP-INSERT(A, x, n)
if A.heap-size == n
        error "heap overflow"
A.heap-size = A.heap-size + 1
k = x.key
x.key = −∞
A[A.heap-size] = x
map x to index heap-size in the array
MAX-HEAP-INCREASE-KEY(A, x, k)
```

Priority Queue
Example

- #include <queue>
  - priority_queue<pair<int, int>> pq;
    - ENQUEUE: pq.push
    - DEQUQUE: pq.pop

Meeting Room Allocation Problem

meeting.cpp

- **Given a list of intervals representing meeting times, find the minimum number of meeting rooms required.**
  - Key Idea: Manage room allocation efficiently with a priority queue of end times.

```
intervals = {{0, 30}, {5, 10}, {15, 20}}
```

```
Minimum number of meeting rooms required: 2
```