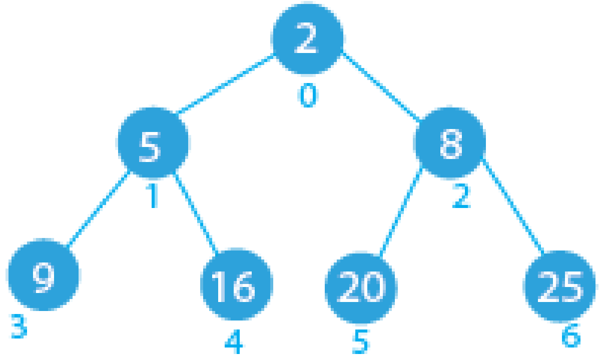
SC1007
Heap

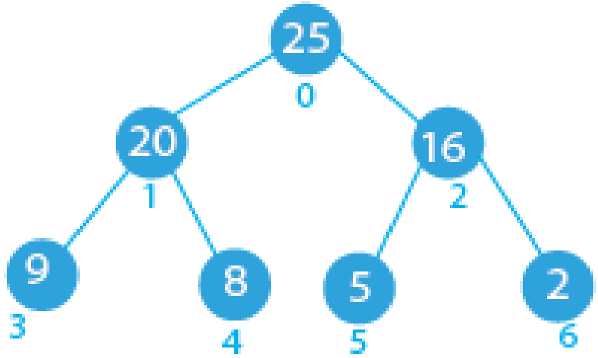
Dr Liu Siyuan
Email: syliu@ntu.edu.sg
Office: N4-02C-72a

What is a Heap

- A binary tree with two key properties:
  - Complete binary tree:
    - All levels are full, except possibly the last. If the last levels is not fully filled, nodes will be filled from left to right.
  - Two kinds of binary heap:
    - Max-heap: Parent ≥ children
    - Min-heap: Parent ≤ children



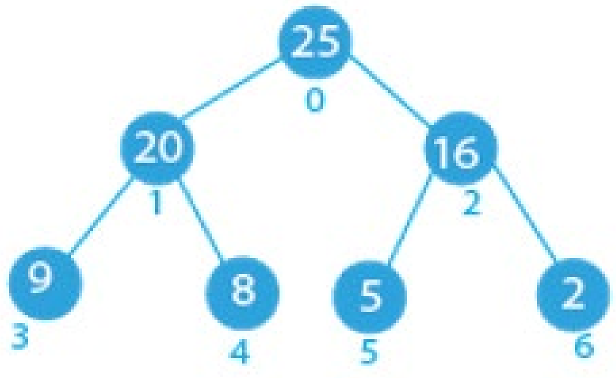
Min-heap



Max-heap

Heap Implementation with an Array

- Heaps can be stored efficiently in an array
- For node at index i:
  - Left child: 2i + 1
  - Right child: 2i + 2
  - Parent: $\left\lfloor \frac{i-1}{2} \right\rfloor$
- The height of a heap is $\lfloor \log_2 n \rfloor$



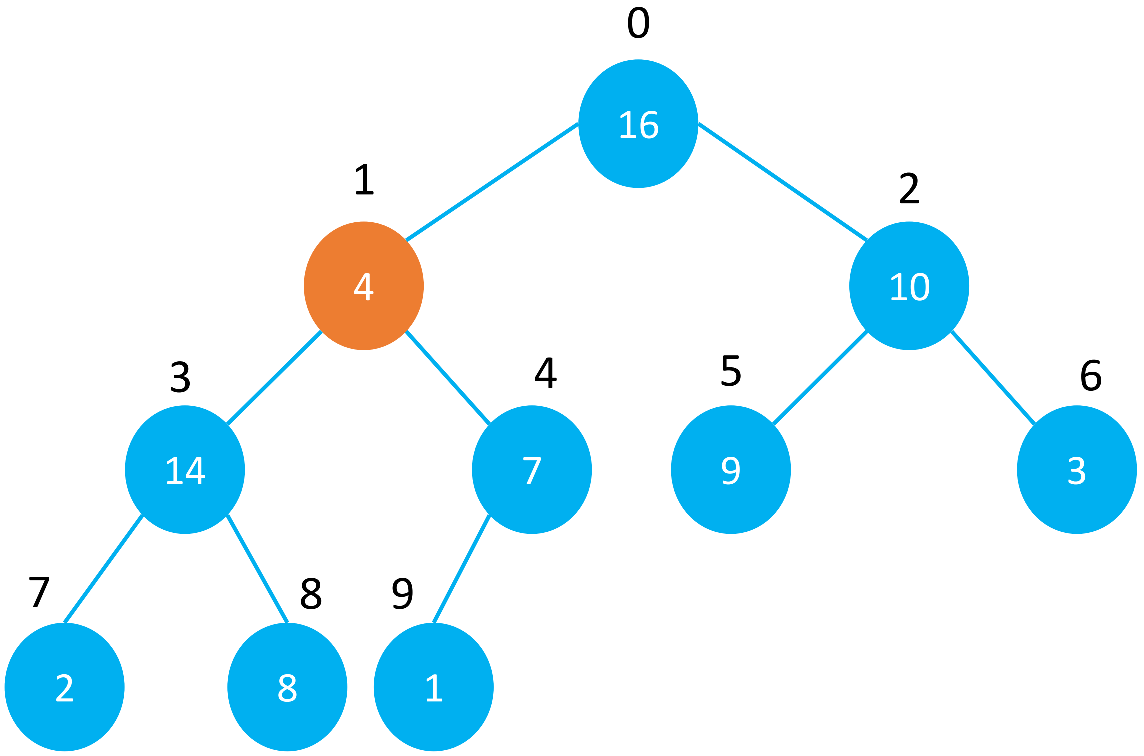
0	1	2	3	4	5	6
25	20	16	9	8	5	2

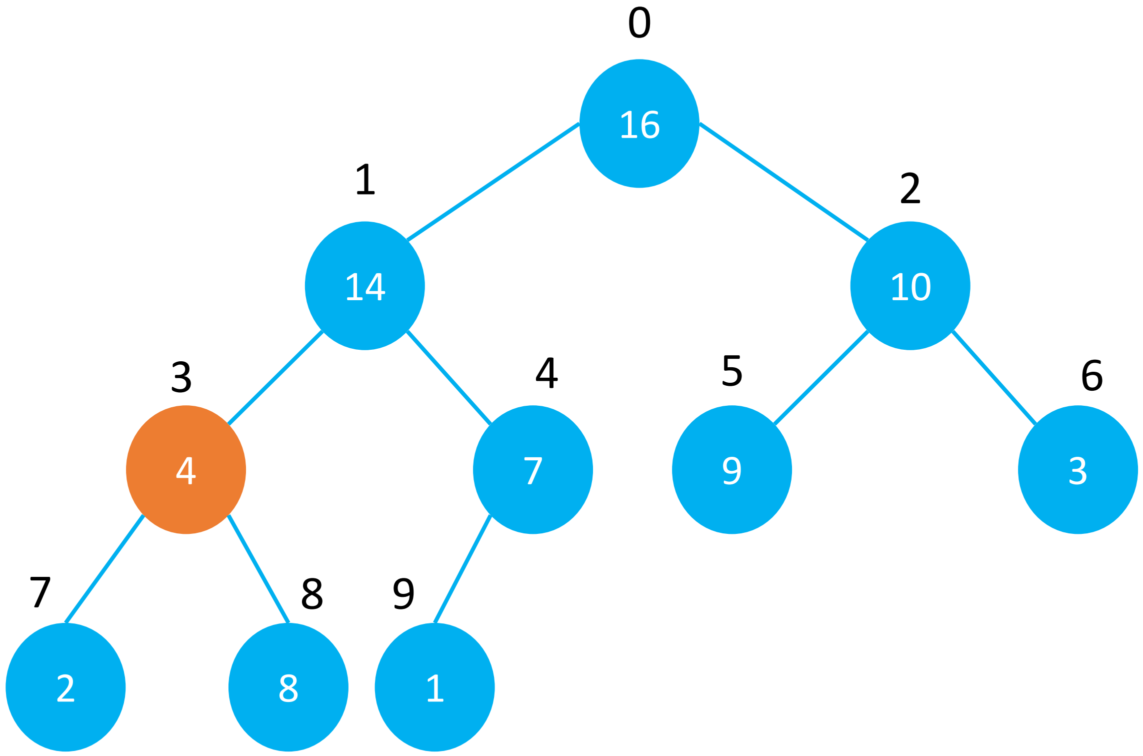
Core Operations -- Heapify

- Given an array A and an index i, assuming that the left and right subtree of A[i] are max-heaps
- A[i] is smaller than its children, violating the max-heap property
- Let the value at A[i] float down in the max heap so that the subtree rooted at index i becomes a max -heap

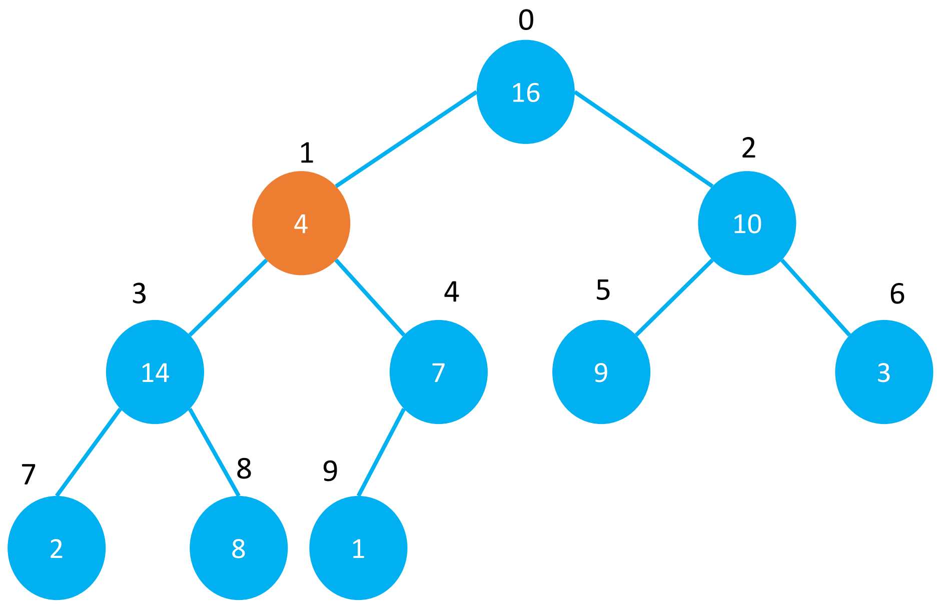
Core Operations -- Heapify

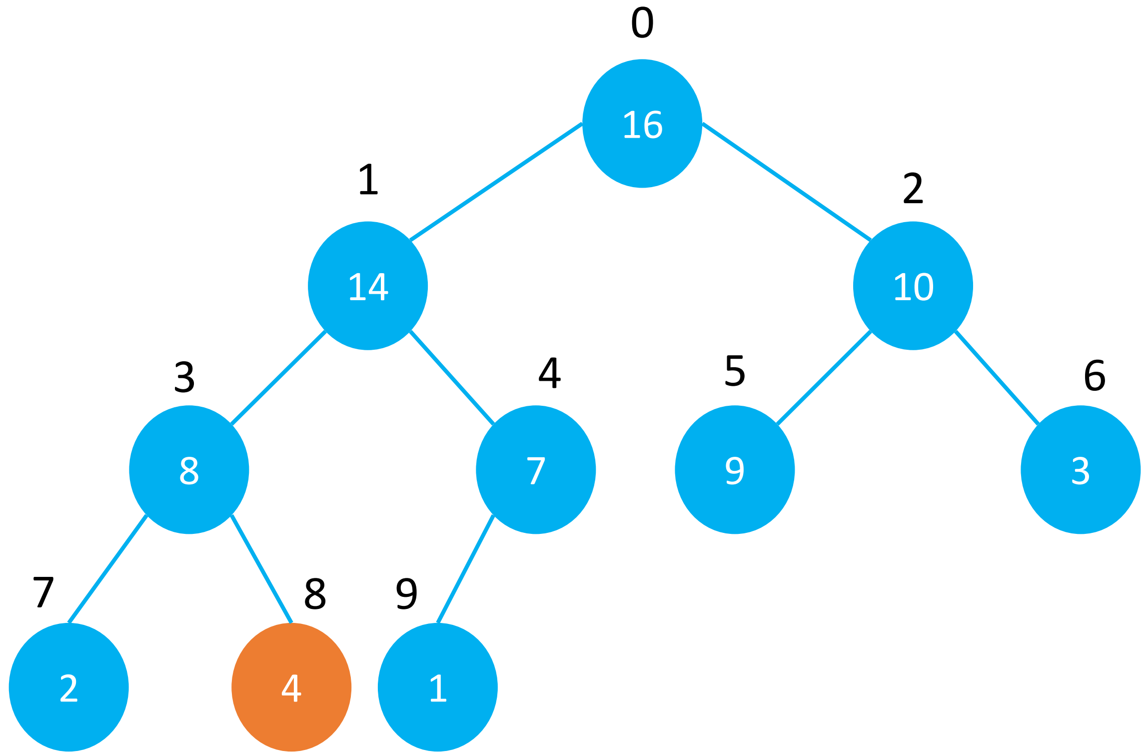
0	1	2	3	4	5	6	7	8	9
16	**4**	**10**	**14**	**7**	**9**	**3**	**2**	**8**	**1**





Core Operations -- Heapify





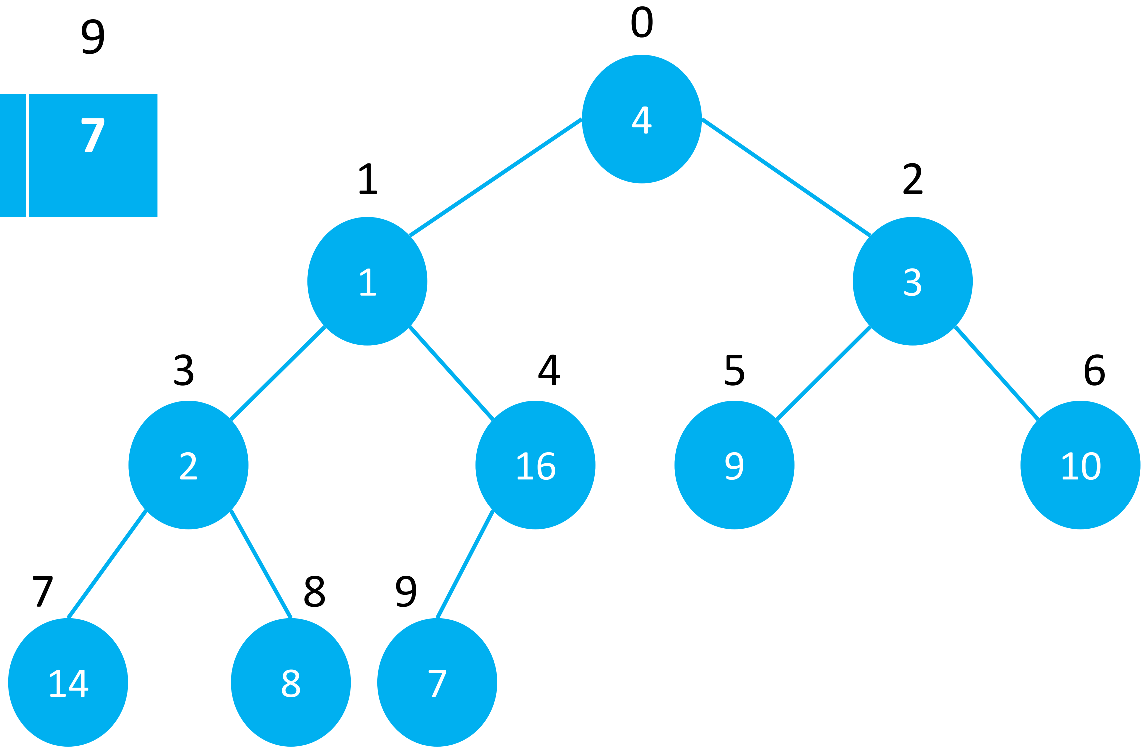

```
MAX-HEAPIFY(A, i)
l = LEFT_CHILD(i)
r = RIGHT_CHILD(i)
if A[l] > A[i]
    largest = l
else largest = i
if A[r]>A[largest]
    largest = r
if largest ≠ i
    exchange A[i], A[largest]
    Max-HEAPIFY(A,largest)
```

The running time of MAX-HEAPIFY on a subtree of size n is O(logn)

Core Operations – Build a Heap

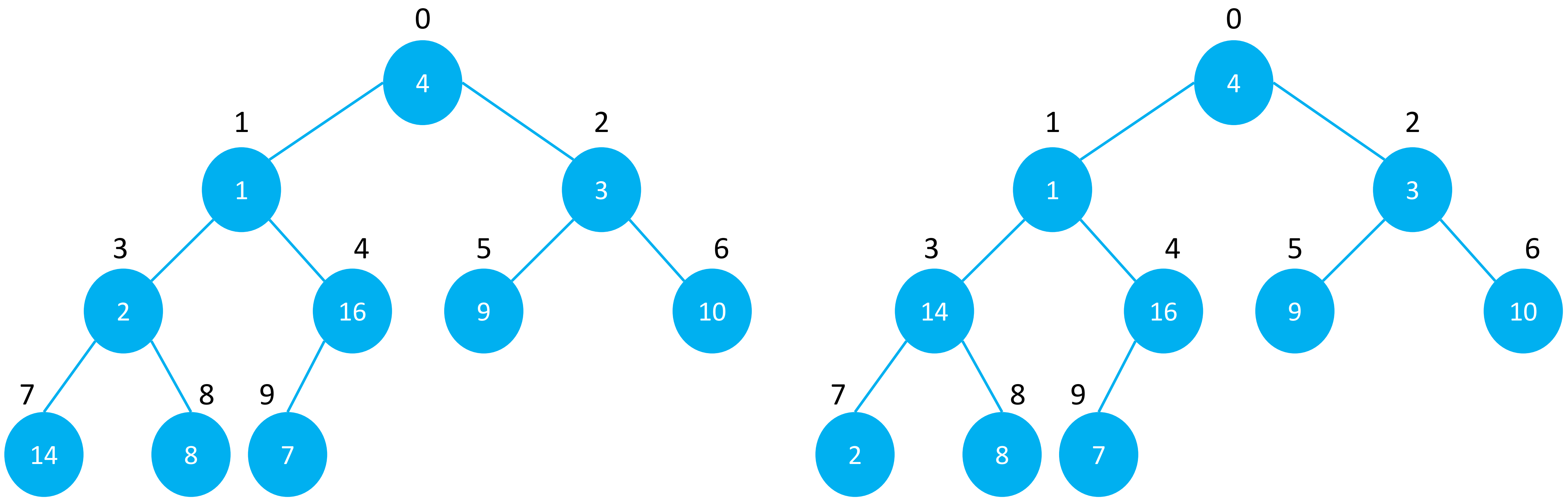
- Convert an array A of length n to a max-heap

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

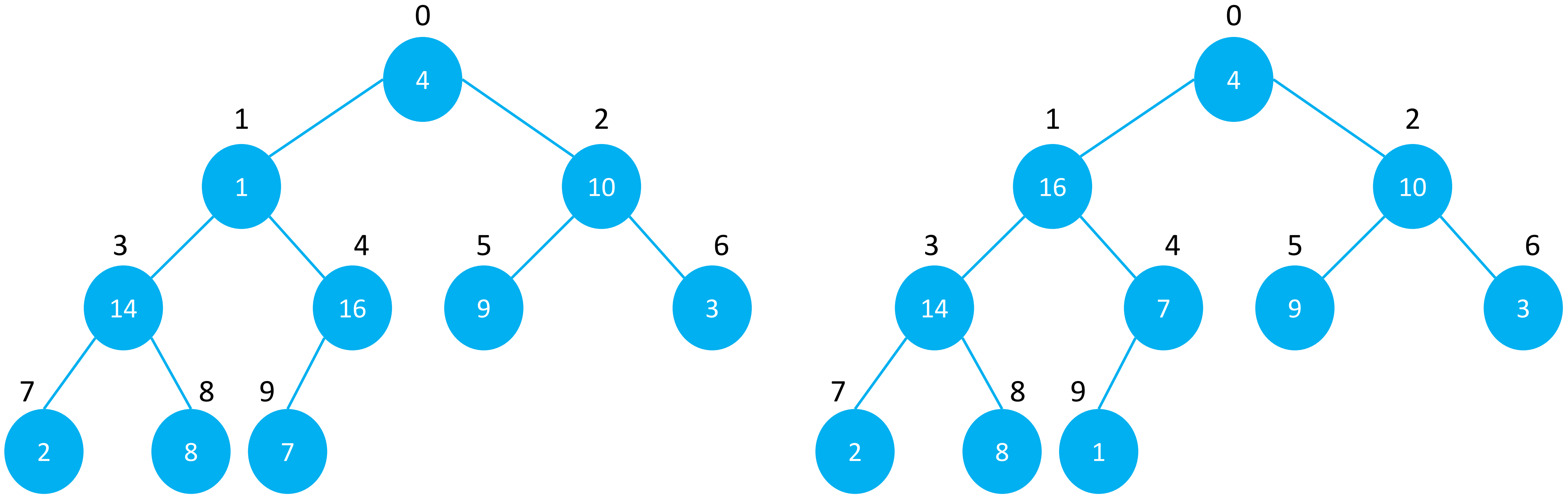



```
BUILD-MAX-HEAP(A)
for i = ⌊(length[A] − 1)/2⌋ to 0
    MAX-HEAPIFY(A,i)
```

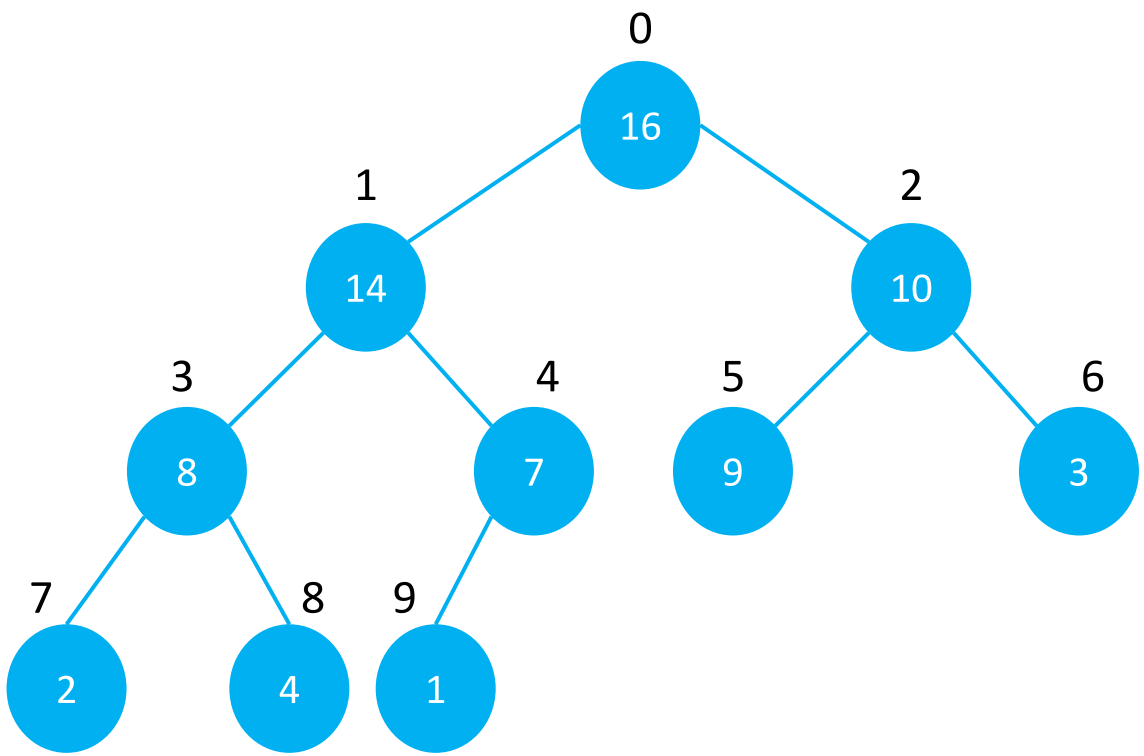
Core Operations – Build a Heap



Core Operations – Build a Heap



Core Operations – Build a Heap



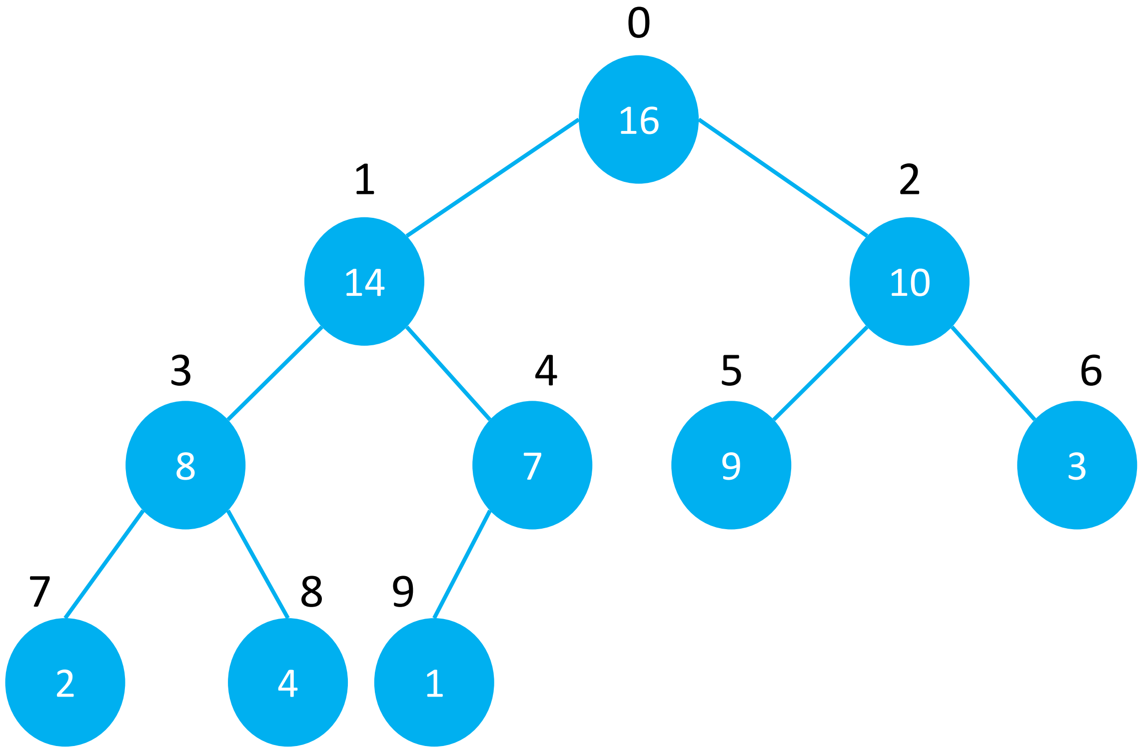
BUILD-MAX-HEAP(A)

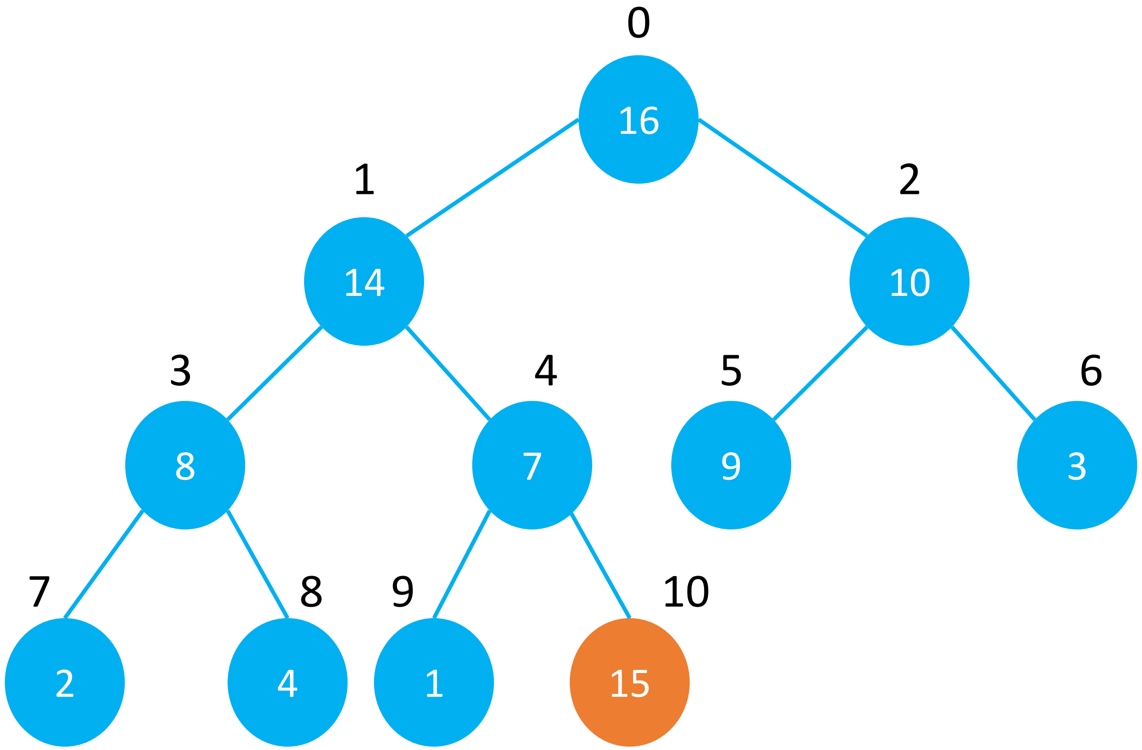
```
for i = ⌊(length[A] − 1)/2⌋ to 1
    MAX-HEAPIFY(A,i)
```

The running time of building a max-heap from an array of size n is O(n)

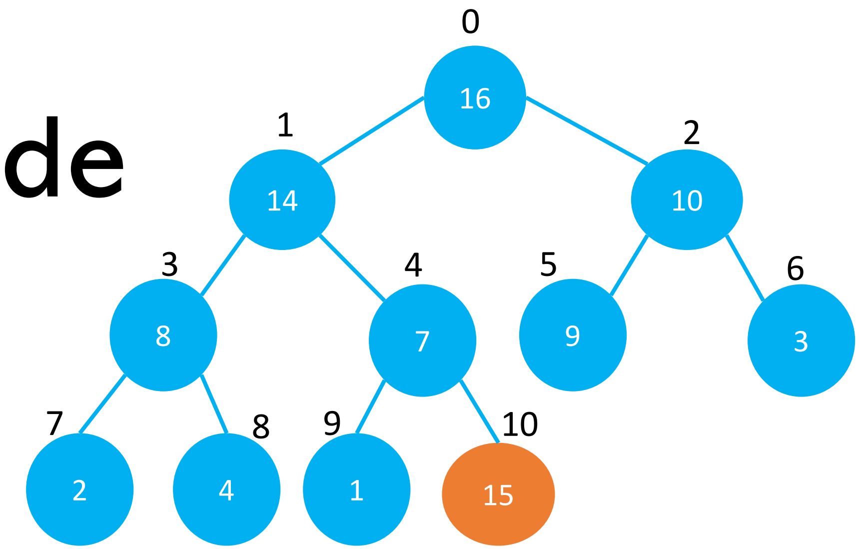
Core Operations – Insert a Node

- Insert into next available slot
- Bubble up until it is heap ordered

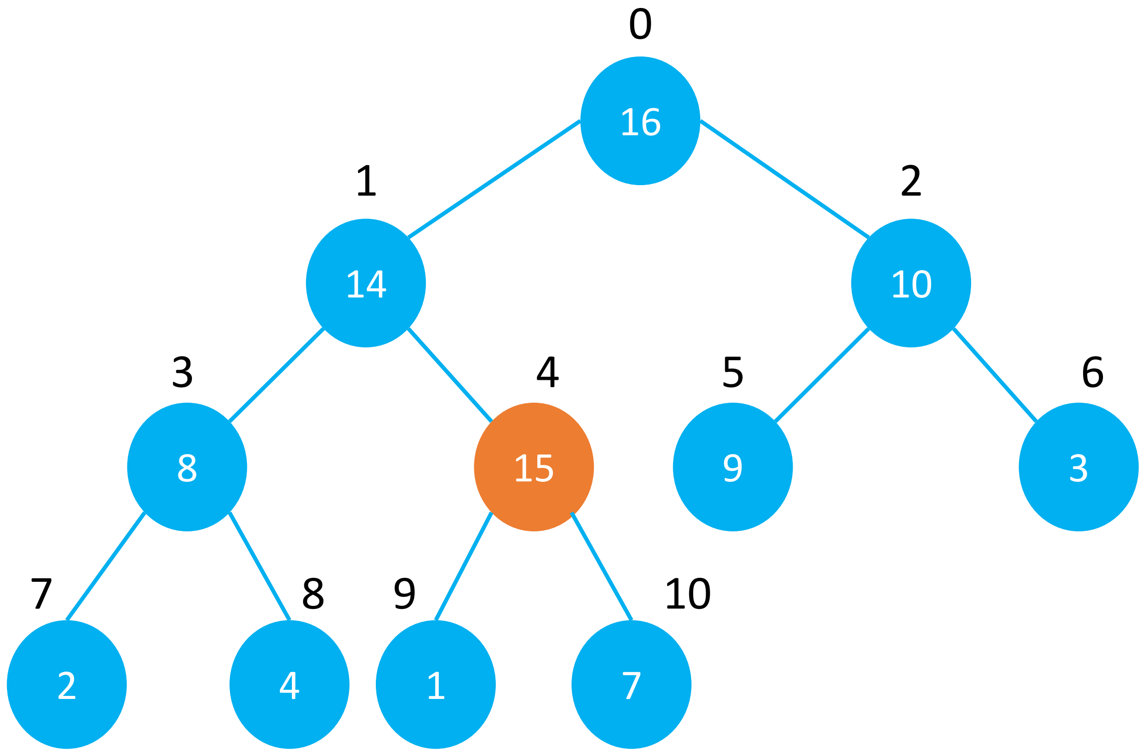


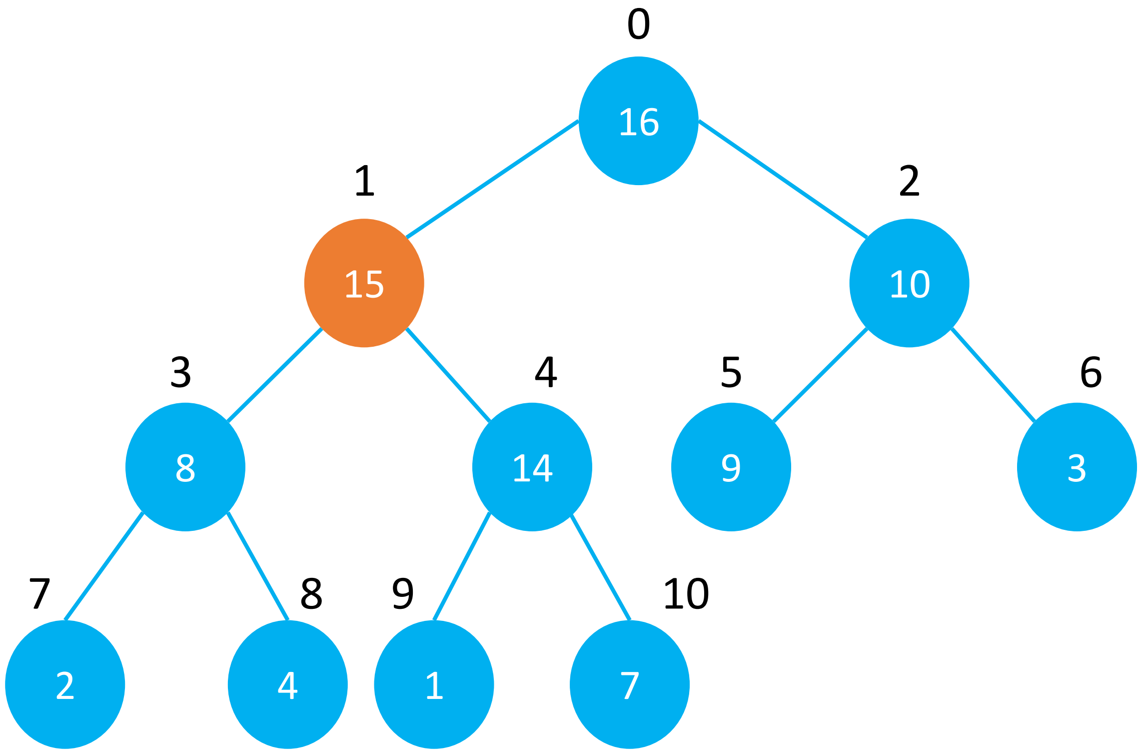


Core Operations – Insert a Node



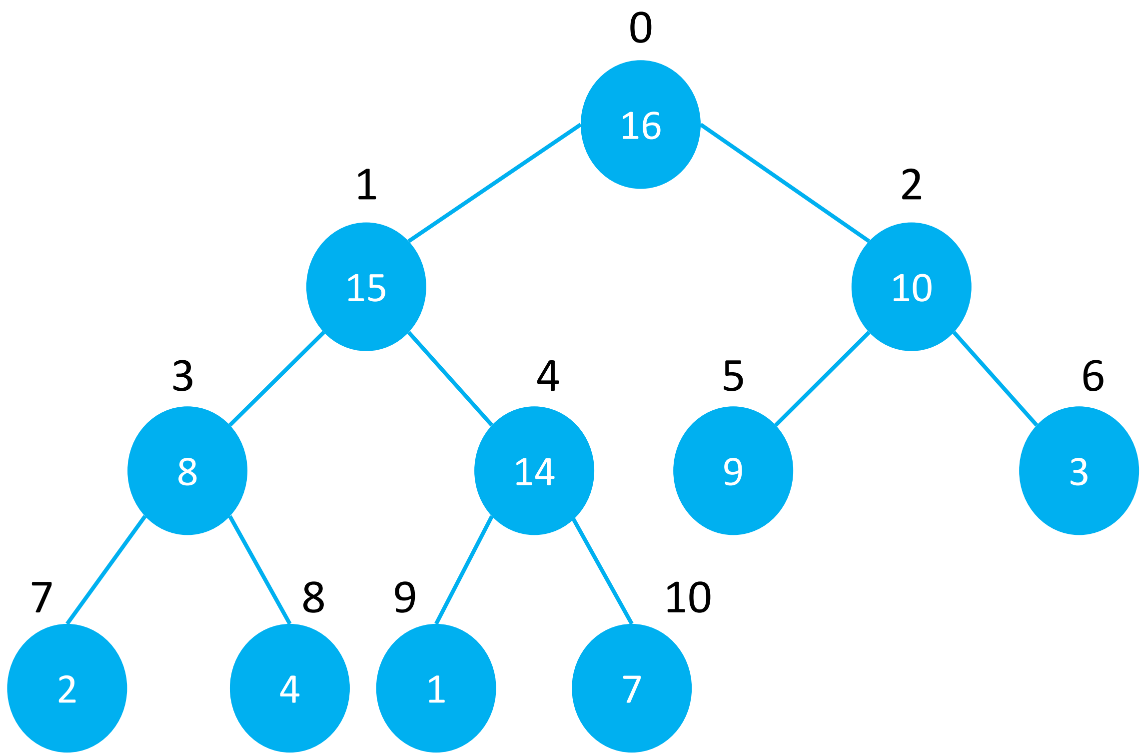
- Insert into next available slot
- Bubble up until it is heap ordered
- The running time to insert a node is O(logn)

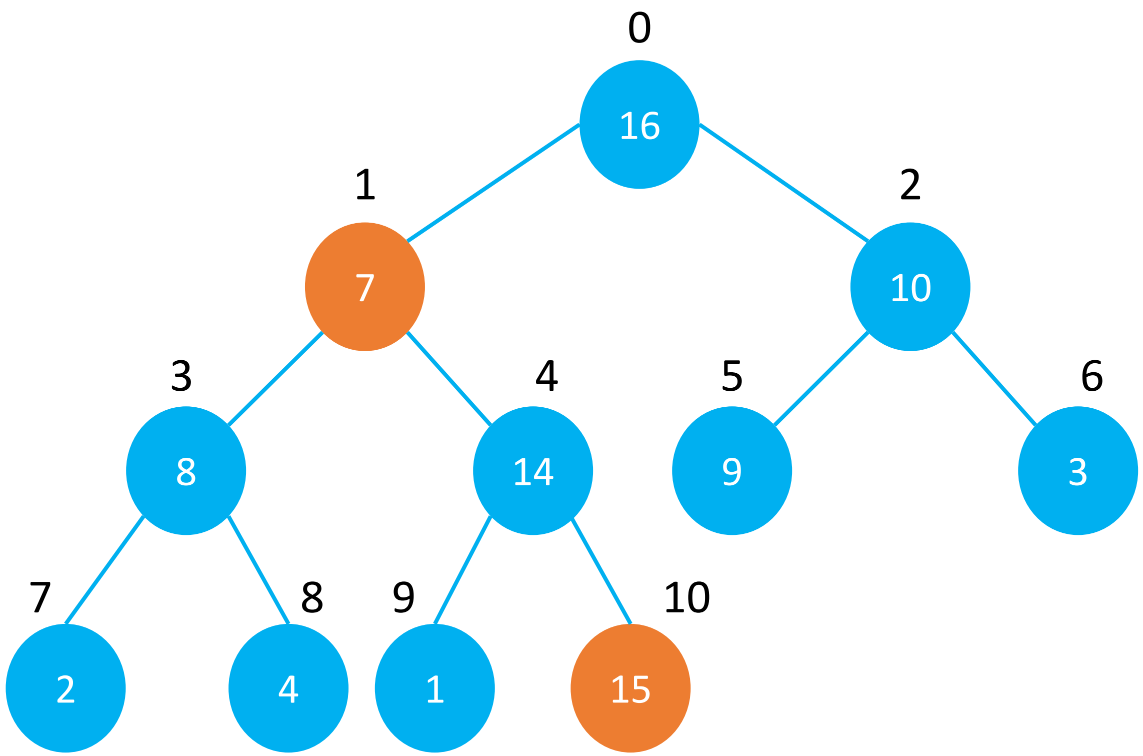




Core Operations – Delete a Node

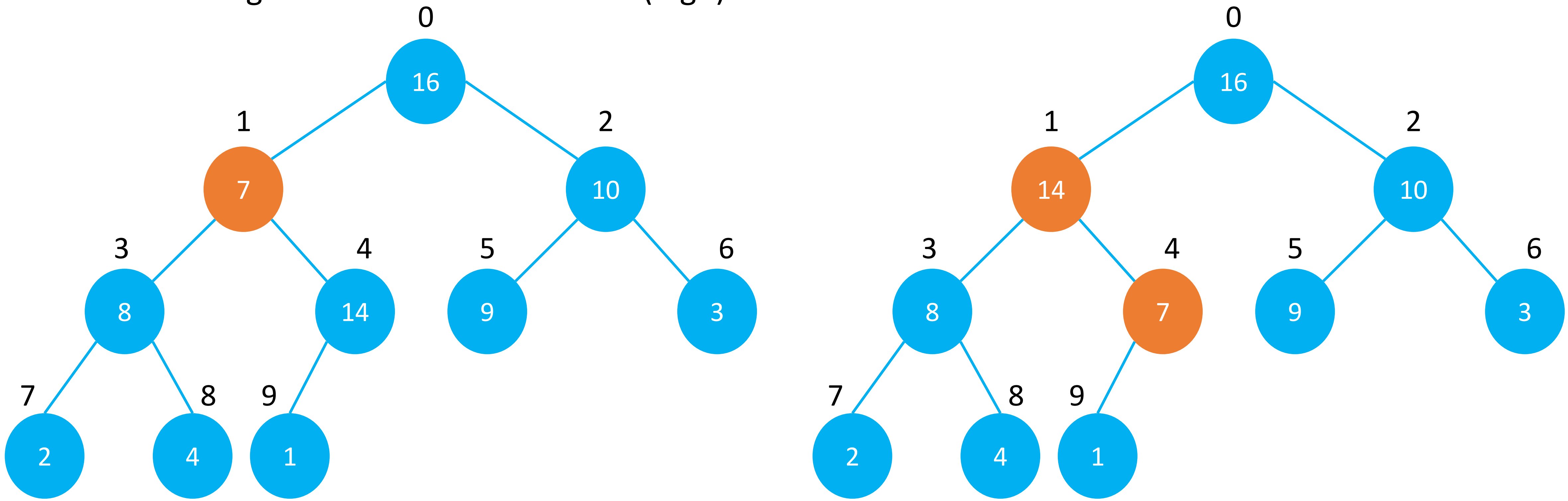
- Replace the node to be deleted with the last element in the heap
- Remove the last element from the heap
- Restore the heap property: If the new value is greater than its parent, bubble up. If the new value is less than one of its children, heapify down





Core Operations – Delete a Node

- Replace the node to be deleted with the last element in the heap
- Remove the last element from the heap
- Restore the heap property: If the new value is greater than its parent, bubble up. If the new value is less than one of its children, heapify down
- The running time to delete a node is O(logn)



Applications of Heap

- Priority Queues
  - Tasks are executed based on priority, not arrival order
  - Example: Operating system task scheduling, print queue
- Heap Sort
  - Efficient sorting algorithm using a heap to repeatedly extract the max/min
  - Time complexity: O(nlog n)
- Graph Algorithms
  - Dijkstra’s Algorithm: Finds shortest path using a min-heap to select the closest unvisited node.
  - Prim’s Algorithm: Builds a minimum spanning tree using a min-heap to pick the cheapest edge.