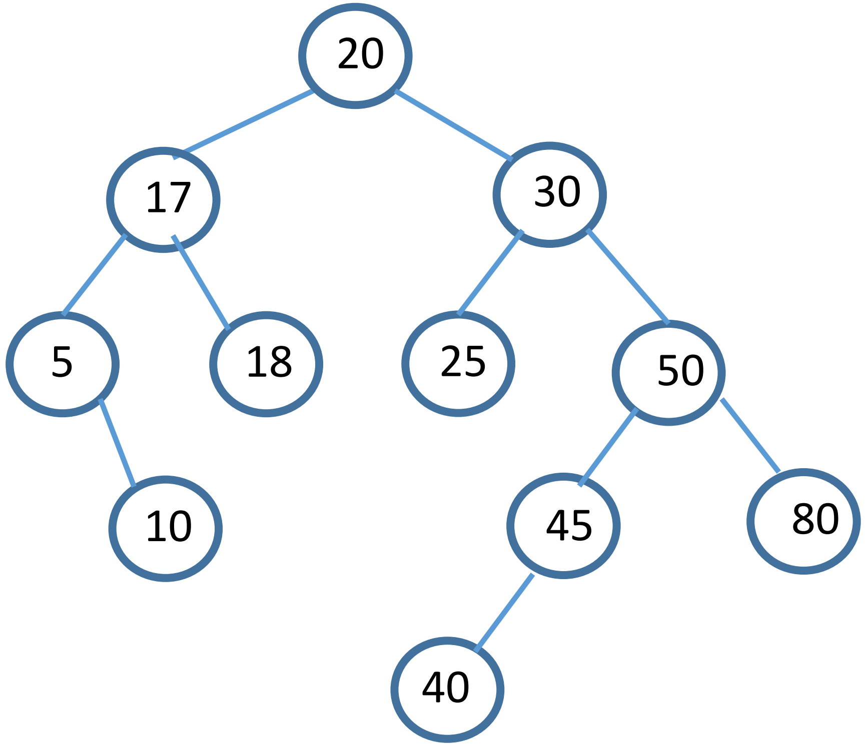
COMP 2611, DATA STRUCTURES
LECTURE 12

BINARY SEARCH TREES

- **Performance Analysis**

HEAPS

Performance Analysis of a Binary Search Tree



- How does it compare to linked list?
- How does it compare to array?

- The height of a binary tree is the maximum depth of any leaf node.

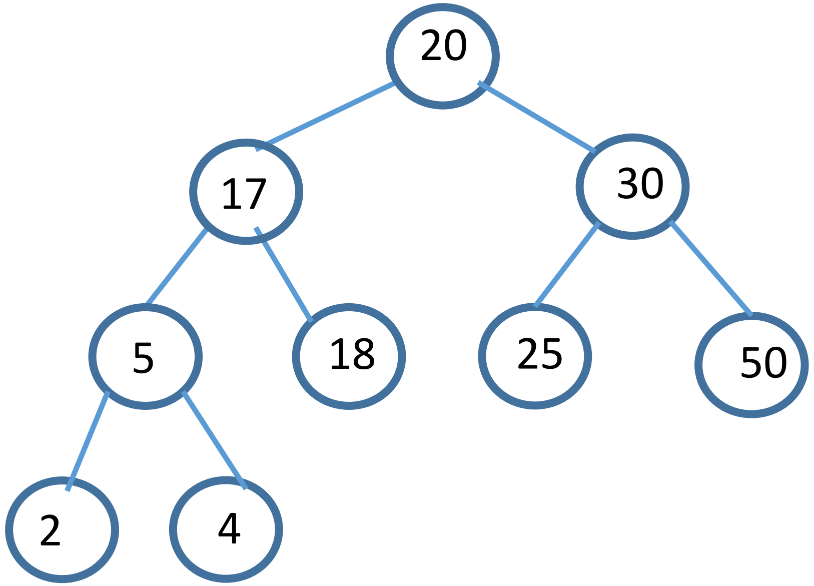
- The height of the BST, *h*, is 4.

- The best case to insert a node is O(*h*). The worst case is O(*n*).
- The best case to search for a node is O(*h*). The worst case is O(*n*).
- The best case to delete a node is O(*h*). The worst case is O(*n*).

The best case for *h* is $\log_2 (n)$

Almost Complete Binary Tree

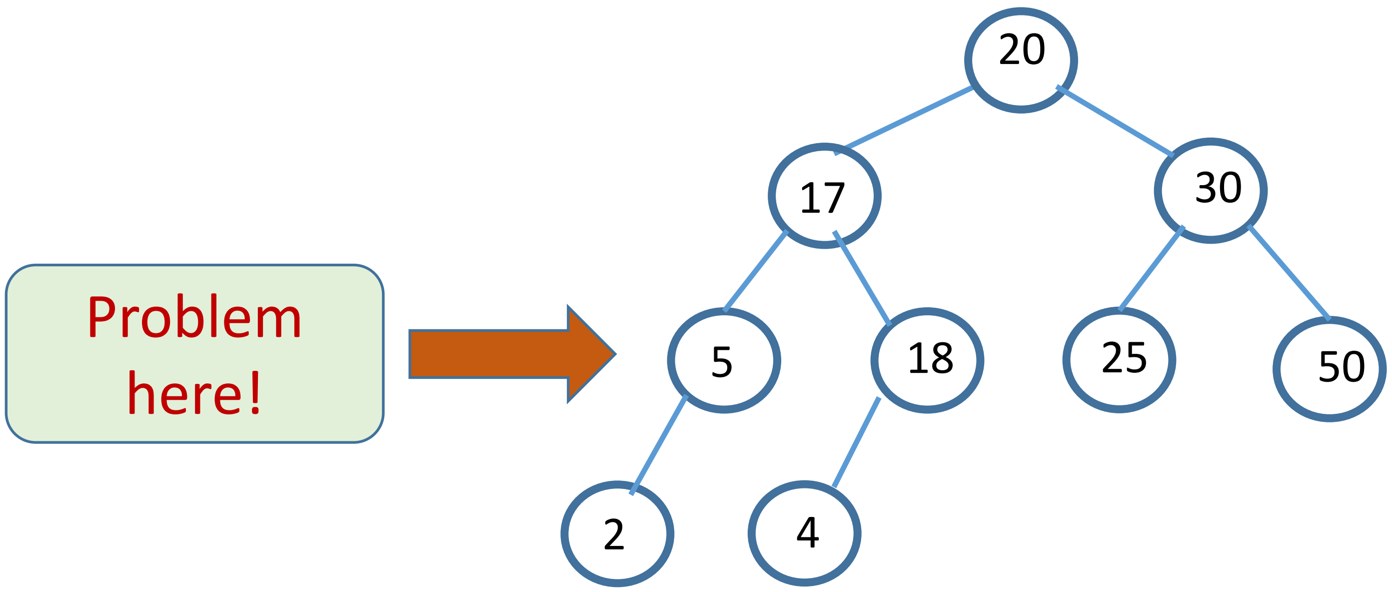
- An *almost complete* binary tree is one in which:
  - ✓ All levels, except possibly the lowest, are completely filled.
  - ✓ The nodes at the lowest level (all leaves) are as far left as possible.



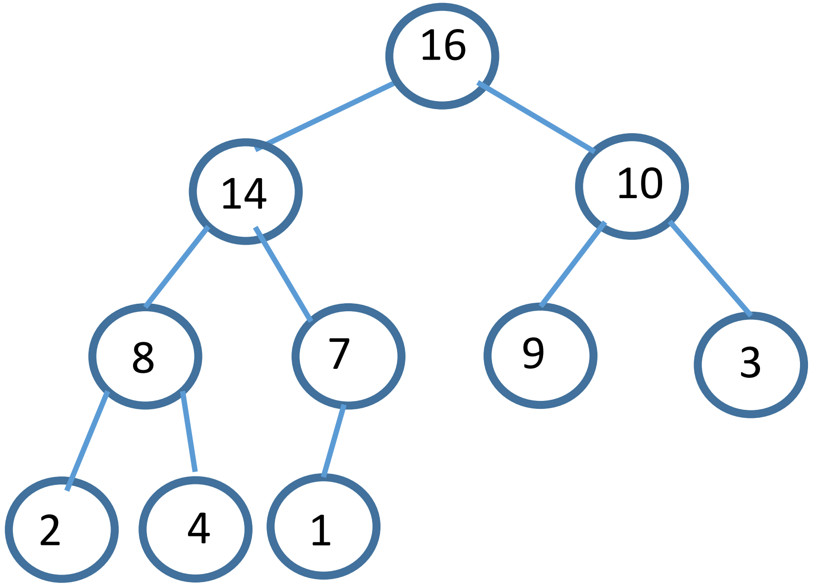
Insert: 20, 17, 30, 5, 18, 25, 50, 2, 4

Almost Complete Binary Tree

- Is the following binary tree almost complete?



Heap: An Almost Complete Binary Tree



- Store the elements of the binary tree in an array:

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

A

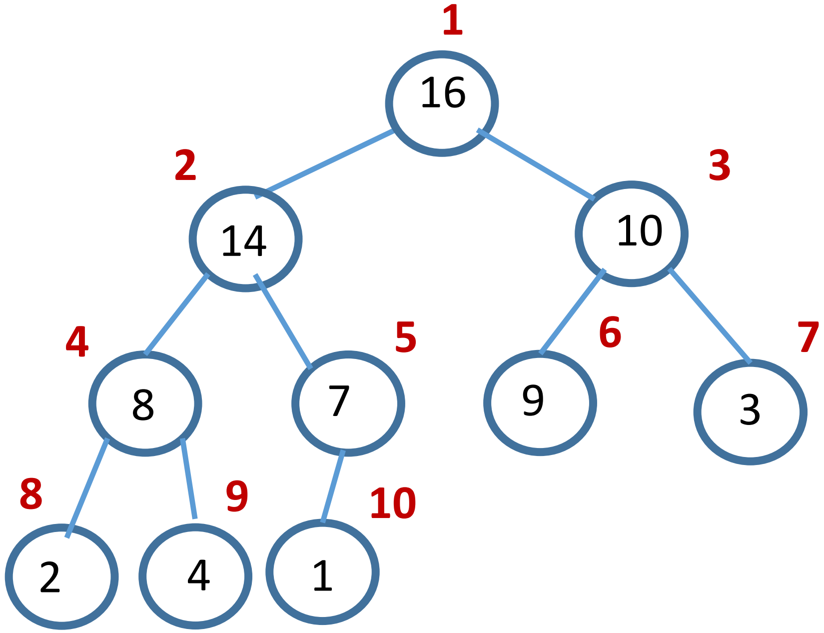
Don't use
location 0

Functions for a Heap

```
Parent (i):
    return floor (i/2)

Left (i):
    return 2*i

Right (i):
    return 2*i + 1
```



- *i* is the index of a node in the array:

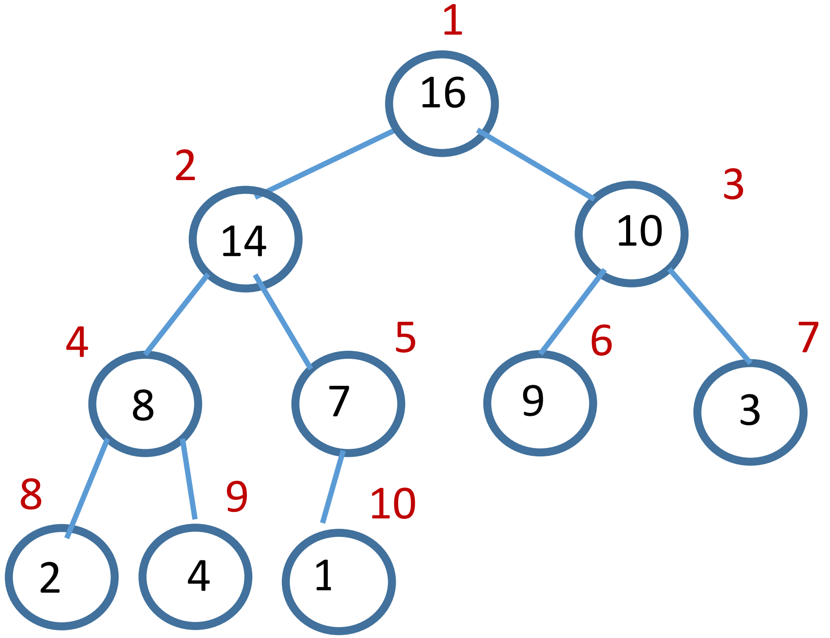
0	1	2	3	4	5	6	7	8	9	10
	16	14	10	8	7	9	3	2	4	1

location 0 is
not used

A Max-Heap

A max-heap satisfies the max-heap property:

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

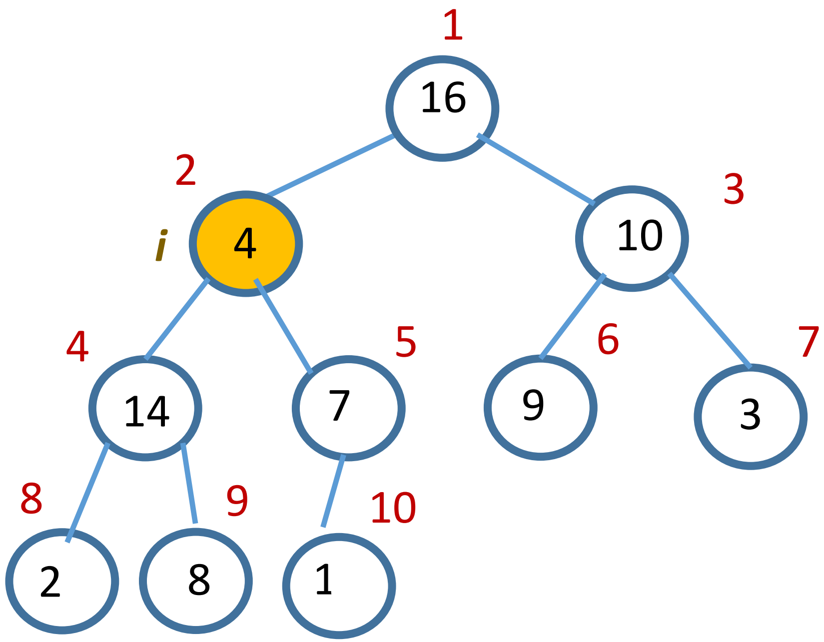


0	1	2	3	4	5	6	7	8	9	10
	16	14	10	8	7	9	3	2	4	1

A

Maintaining the Max-Heap Property

- Suppose we know that:
  - The binary trees rooted at Left (*i*) and Right (*i*) are max-heaps, but,
  - *A*[*i*] might be smaller than its children.

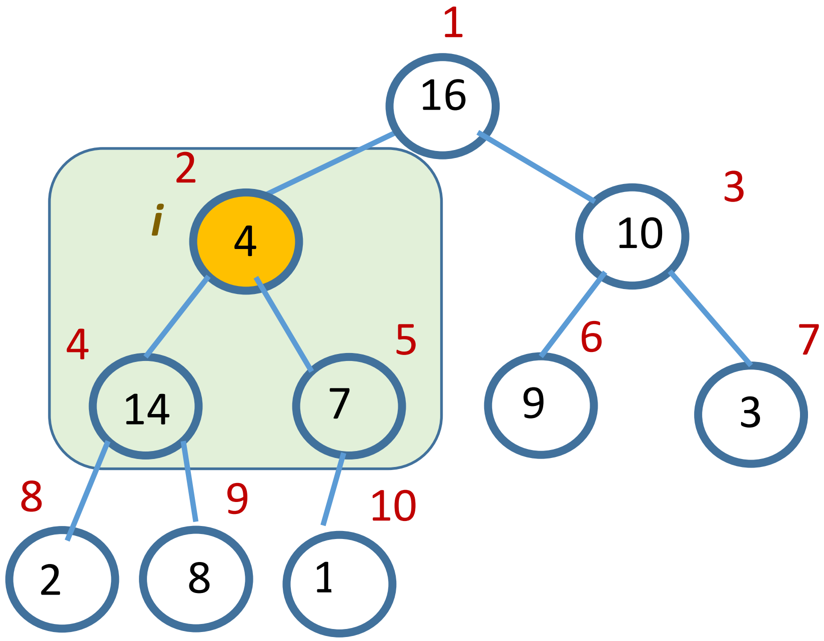


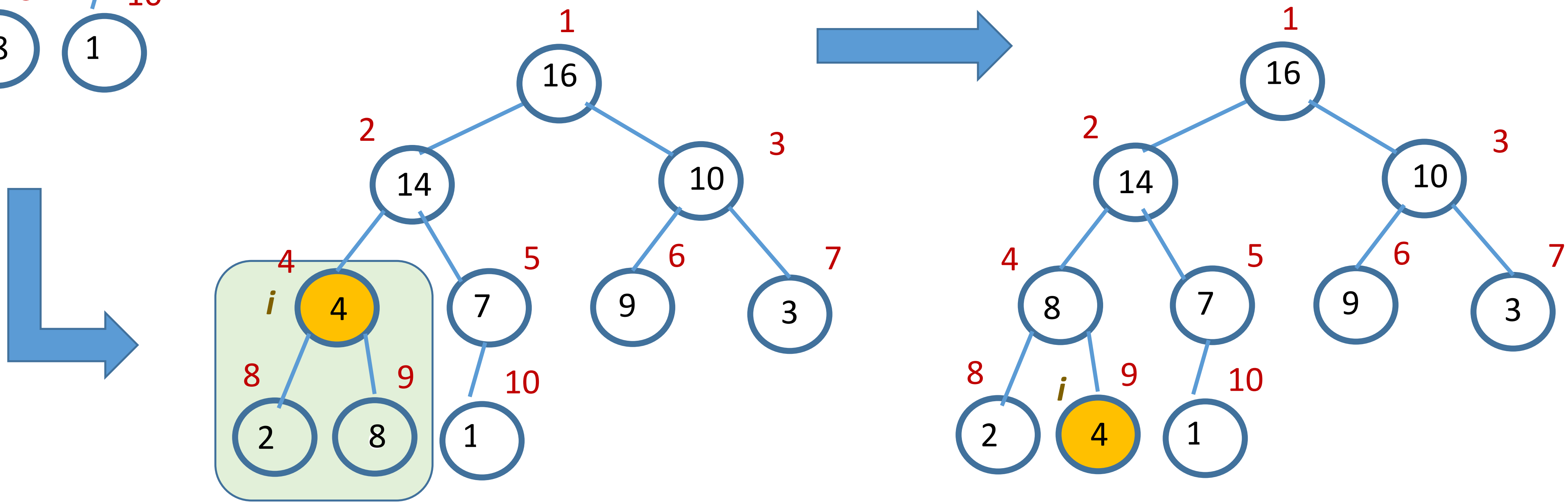
- How can we maintain the max-heap property?

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

A

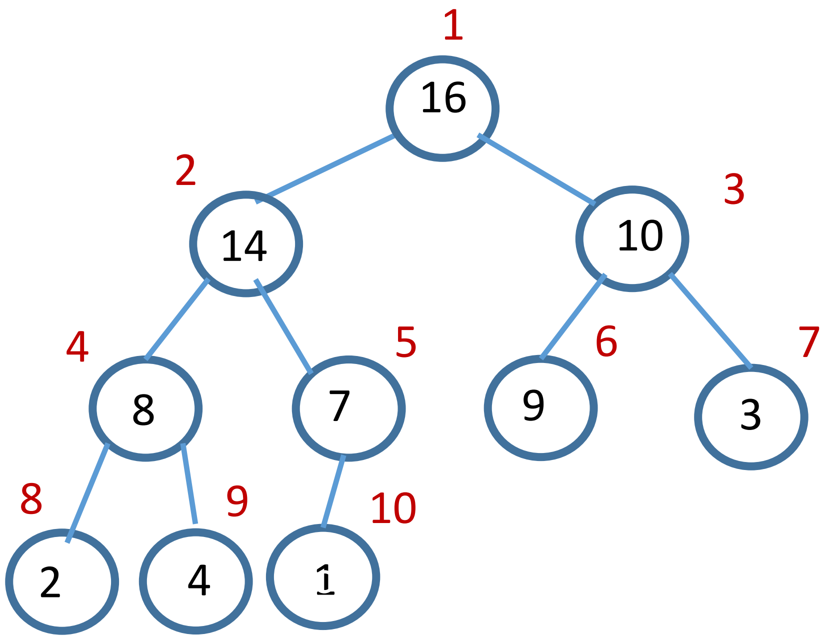
Maintaining the Max-Heap Property





We can write a function, **maxHeapify** (*A*, *i*) to restore the binary tree to a max-heap.

Declaration of a Max-Heap




```
struct MaxHeap {
    int A [1000];
    int size;
};

MaxHeap * heap;
```

- *i* is the index of a node in the array:

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

location 0 is not used

The *maxHeapify* Function

```
maxHeapify (MaxHeap * heap, int i) {

      left = i * 2;
      right = i * 2 + 1;

      largest = index of largest of:
                          heap->A[i],
                          heap->A[left],
                          heap->A[right]


      if (largest != i) {
            swap heap->A[largest] with heap->A[i];
            maxHeapify(heap, largest);
      }
}
```