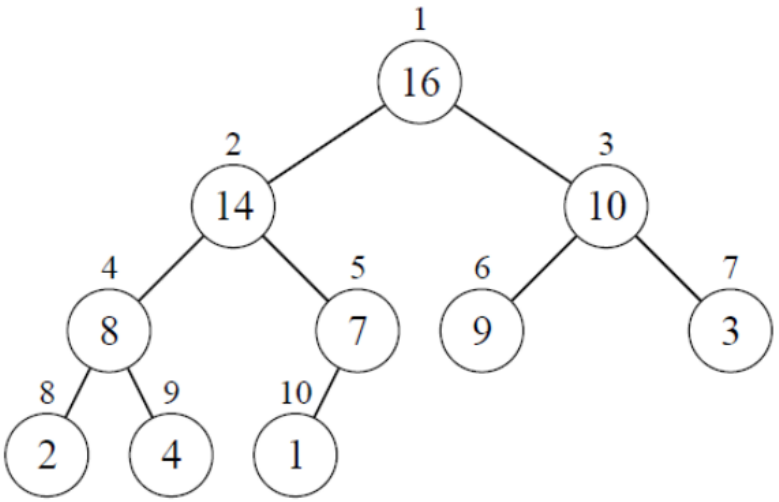
HEAPS

CS340

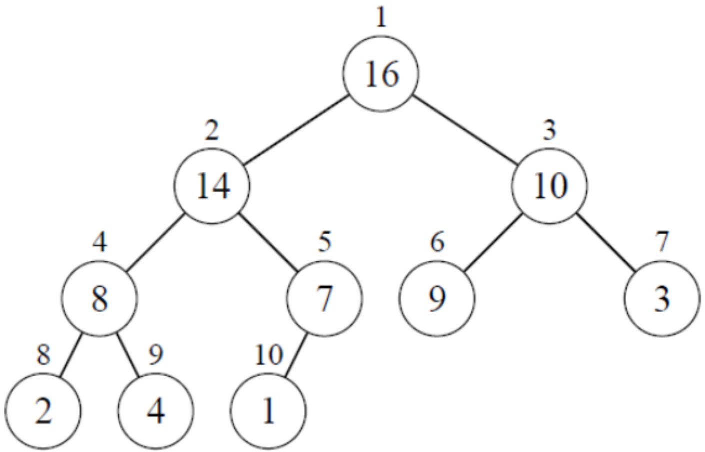
Properties of a heap

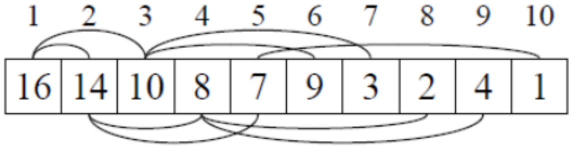


- A nearly-complete binary tree.
- Max-heap or min-heap.
- Largest element is at the root of a max-heap.
- **The Heap Property (max heap):**
  Parent node ≥ its children
- Recursive definition:
  Each node is the root of its own heap.
- Height of node = # of **edges** from the node down to a leaf.
- Height of heap = height of root = Θ(lg n).

Heap stored as array

- **Root** = max value = A[1]
- **Parent** of A[i] = A[i/2]
- **Left child** of A[i] = A[2i]; **Right child** of A[i] = A[2i+1]
- Computing is fast with binary representation





Heapsort basic methods

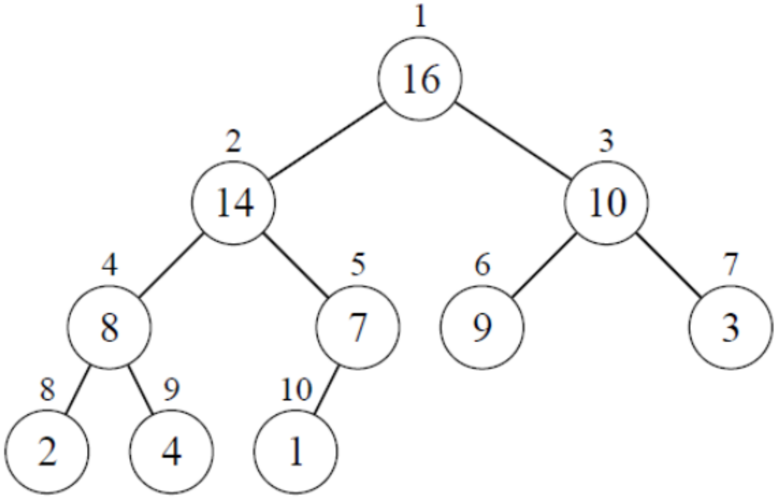
```
int parent(int i) {

}
int left(int i) {

}
int right(int i) {

}
```

In a max-heap, `A[parent(i)] ≥ A[i]`



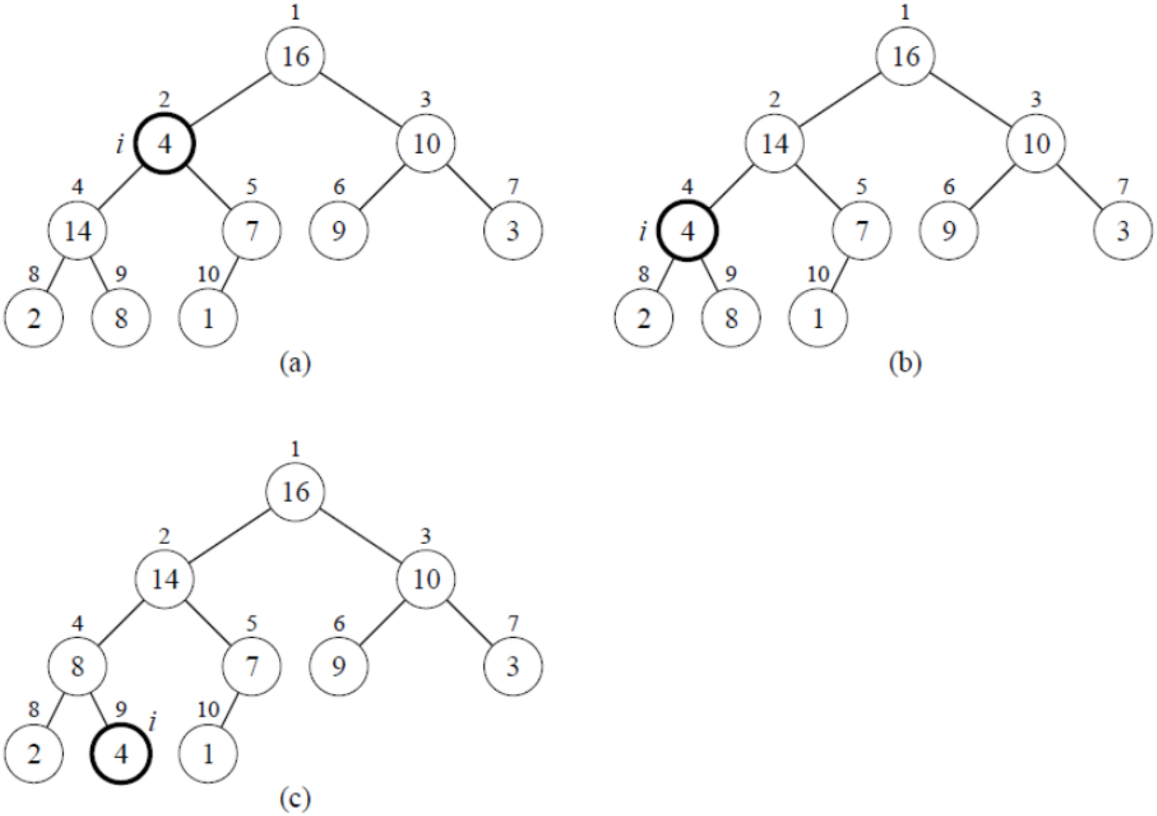
Maintaining the heap property

- Check that A[i] is larger than its children
- If not exchange it with its larger child, and recursively call MAX-HEAPIFY
- lets the value at A[i] “float down” in the max-heap
- Makes node i a max-heap root

```
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)
```

Max-Heapify Example

- Node 2 violates heap property, is switched with node 4.
- Then, node 4 violates heap property



Running time of Max-Heapify

- $O(h)$ on a node of height h
- $O(\lg n)$

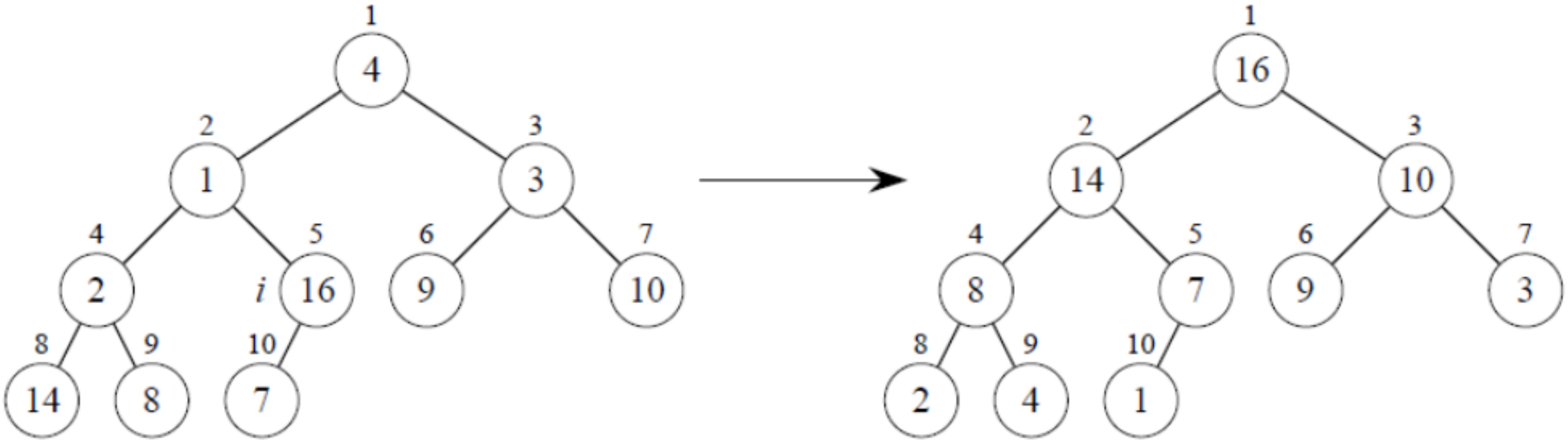
Building a heap

- Starting from an unordered array, build a heap.
- Why does it operate on A.length/2 downto 1?
- Running time seems to be O(n lg n) but it's really **O(n)**
  - Observe that the time for MAX-HEAPIFY to run at a node varies with the height of the node in the tree, and the heights of most nodes are small.

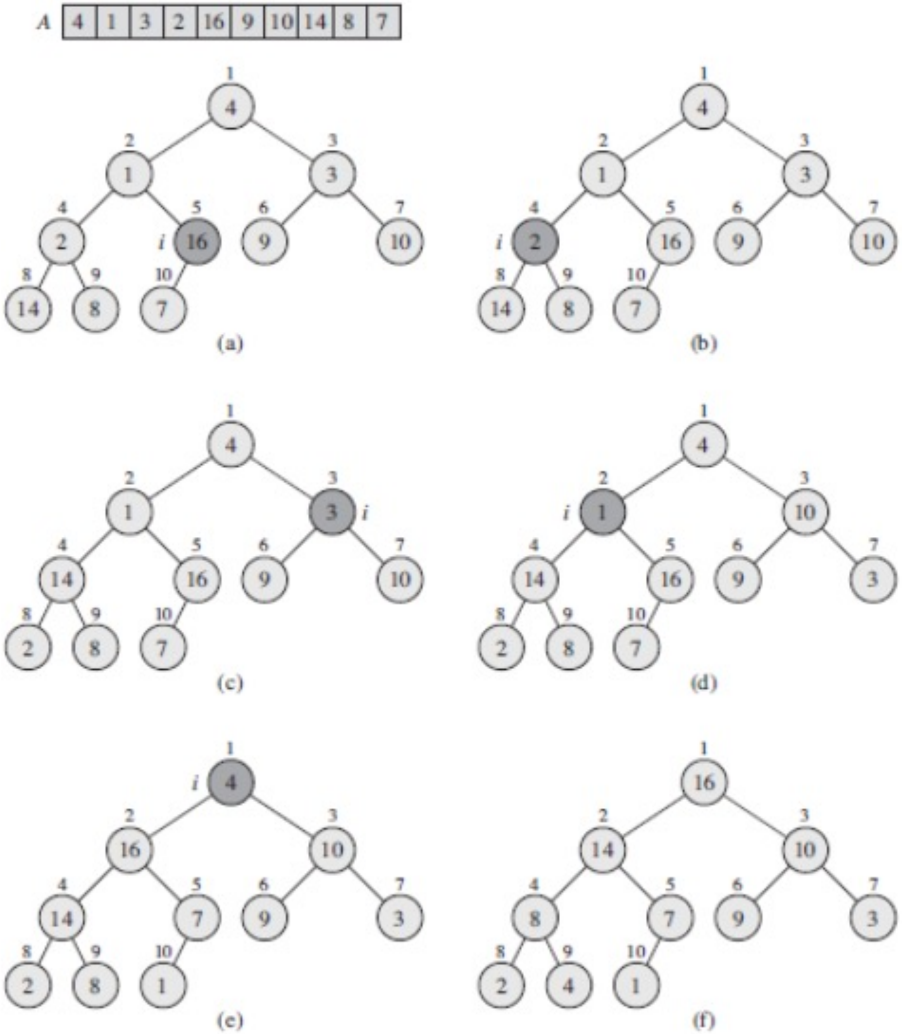
```
BUILD-MAX-HEAP(A)
1  A.heap-size = A.length
2  for i = ⌊A.length/2⌋ downto 1
3      MAX-HEAPIFY(A, i)
```

Build-Max-Heap example

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



Build-Max-Heap example



Build-Heap loop invariant

At the start of each iteration of the for loop, each node i + 1, i + 2, … ,n is the root of a max-heap.

- **Initialization:**, *i = n/2*. Each node *n/2+1, n/2+2, … n* is a leaf and is trivially a max-heap.
- **Maintenance:** To see that each iteration maintains the loop invariant, observe that
- the children of node i are numbered higher than i . By the loop invariant, therefore,
- they are both roots of max-heaps. This is precisely the condition required
- for the call MAX-HEAPIFY(A, i) to make node i a max-heap root. Moreover,
- the MAX-HEAPIFY call preserves the property that nodes i+1, i+2,2,…,n
- are all roots of max-heaps. Decrementing i in the for loop update reestablishes
- the loop invariant for the next iteration.
- **Termination:** At termination, i=0. By the loop invariant, each node 1;2; : : : ;n
- is the root of a max-heap. In particular, node 1 is.