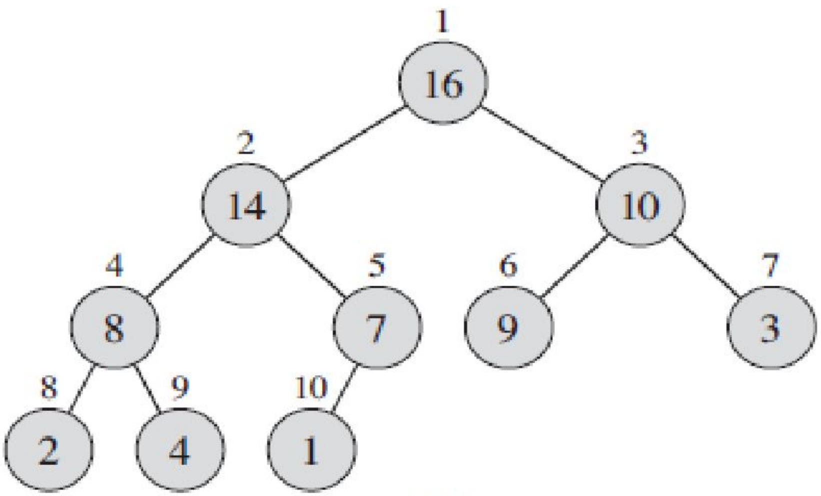
Heapsort Analysis

Nearly complete binary tree

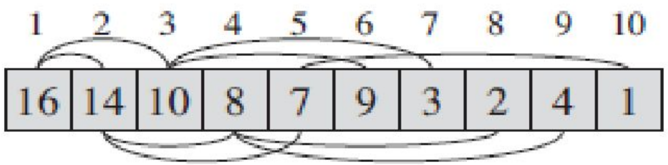
Recall that Heap is viewed as a Nearly complete Binary tree

Each node in the tree – an element of the array

Tree is completely filled on all levels except possibly the lowest, which is filled from the left up to a point



(a)



(b)

Two attributes of an array A

- **A.length:** Number of elements in the array
- **A.heapsize**: How many elements in the heap are stored within array A
- Although **A[1…A.*length]*** may contain numbers, only the elements in **A[1…A.*heapsize]***, where **0<= A.*heapsize <=* A.*length*,** are valid elements of the heap.

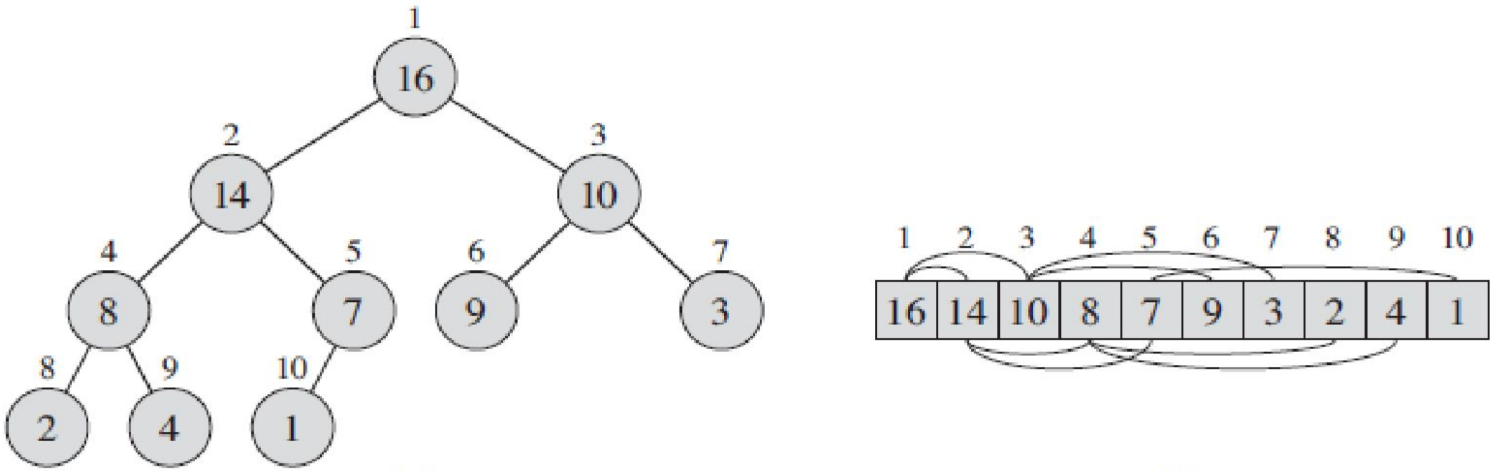
Parent, left and right child – Binary heap

- The **root of the tree is A[1]**, and **given the index *i*** of a node,

  index of its parent, **PARENT(i)**: **floor(i/2)**

  index of left child, **LEFT(i)**: **2*i**

  index of right child, **RIGHT(i)**: **2*i + 1**



(a) (b)

Max and Min Heaps

- There are **two kinds** of binary heaps:
  - max-heaps and
  - min-heaps.
- In both kinds, the values in the nodes satisfy a ***heap property***, the specifics of which depend on the kind of heap.

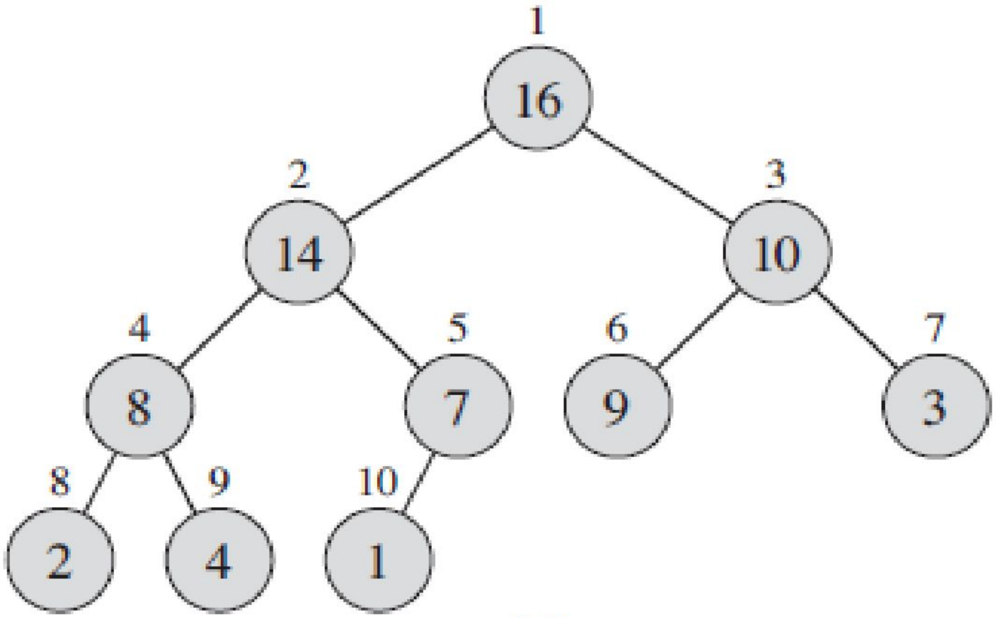
Max-heap

- ***Max-heap property*** is that for every node ***i*** other than the root,

  A[PARENT(i)]>= A[i]

  Value of a node is at most the value of its parent.
- **Largest element** in a **max-heap** is stored at **the root**
- subtree rooted at a node contains values no larger than that contained at the node itself.

Example – Max Heap



(a)

Recap: Notations of a binary heap

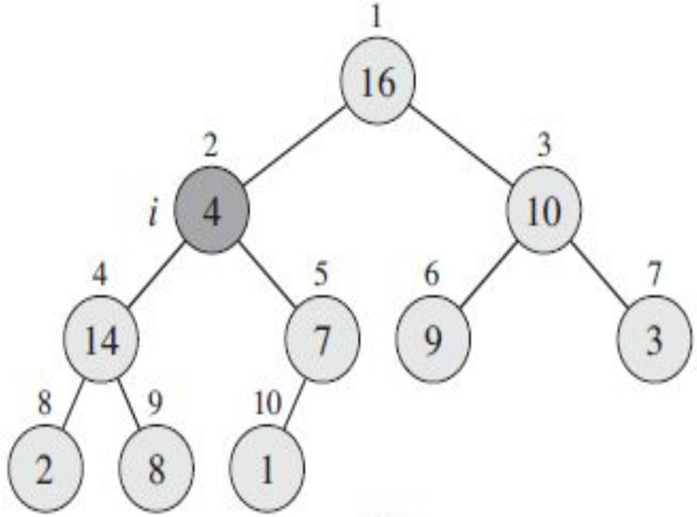
- ***Height* of a node** in a heap: number of edges on the longest simple downward path from the node to a leaf
- **Height of the heap:** height of its root
- Since a heap of **n** elements is based on a complete binary tree, its height is -------------?
- **Θ(lg n)**

How do you establish heap property (Max/Min) in the given input array?

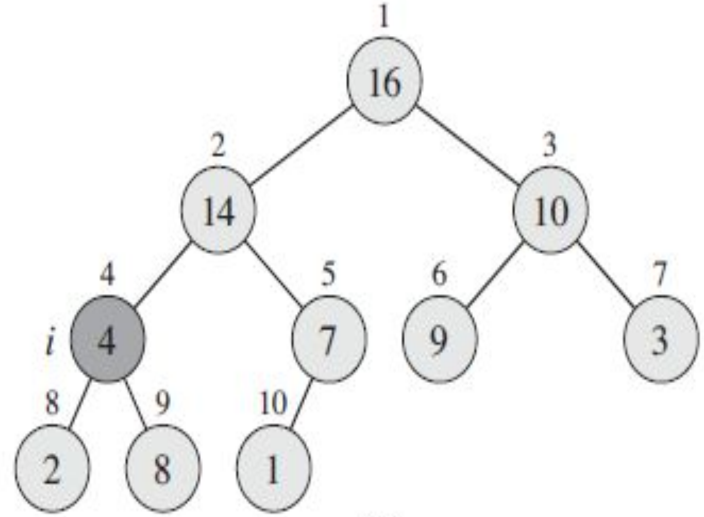
– we apply MAX-HEAPIFY procedure to establish MAX-HEAP property on the ith element of an array, in which the property is violated

Maintaining the heap property

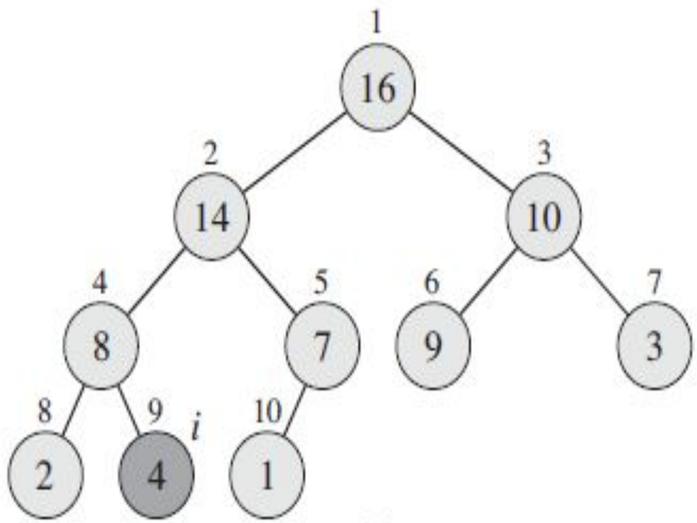
- To **maintain the max-heap property**, use the procedure MAX-HEAPIFY.
- Inputs are **an array A and an index i** into the array.
- MAXHEAPIFY assumes that the binary trees rooted at **LEFT(i) and RIGHT(i) are max heaps**, but that **A[i] might be smaller than its children**
  - violating the max-heap property.
- **MAX-HEAPIFY** lets the value at A[i] “float down” in the max-heap so that the **subtree rooted at index i** satisfies the max-heap property


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

(a)


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

(b)


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

(c)

MAX-HEAPIFY

```
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)
```

Running time of Max-Heapify

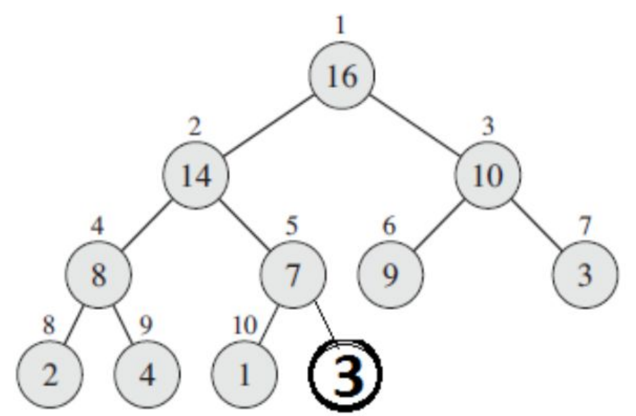
Running time of Max-Heapify =
Running time for (1) & (2)

(1) Time to fix up the relationships between **A[i], A[LEFT(i)] and A[RIGHT(i)]**

(2) Time to run **MAX-HEAPIFY** on a subtree rooted at one of the children of node **i** (assuming that the recursive call occurs).

Running time of **MAX-HEAPIFY** on a **subtree** rooted at one of the **children of node i:**

- Need to know the size of the subtree (# nodes) rooted at one of the children of node i.
- What is the worst case size of the subtree ….?
  - Since heap is a nearly complete binary tree, the worst case occurs when the bottom level of the tree is exactly half full
  - Hence, to find the maximum number of nodes in the left subtree of the nearly complete binary tree



Total number of nodes in a complete binary tree of height h is $2^{h+1} - 1$

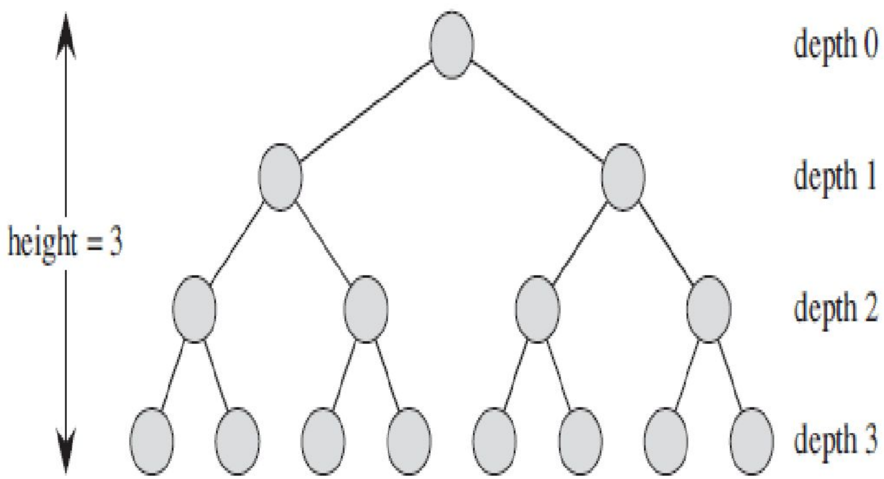
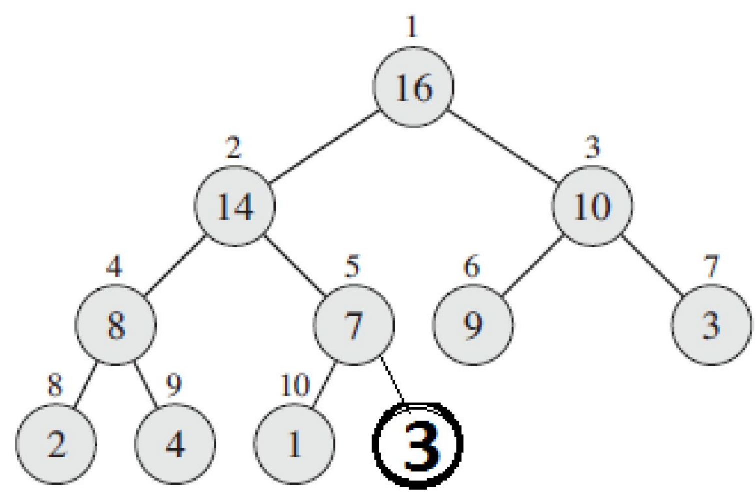


Figure B.8 A complete binary tree of height 3 with 8 leaves and 7 internal nodes.

- Total number of nodes in a nearly complete binary tree of height h is, $(2^{h+1} - 1) - (2^{h-1})$
- Why do we subtract 2^{h-1} from the total number of nodes…..?



- If **h is the height** of the subtree, number of **leaf nodes are 2^h** **(Eg: h=3, #leaves=8)**
- **Since it is a binary tree, $2^h/2$** is the number of leaf nodes in each of the **left and right subtree**

  i.e $2^h/2 = 2^{h-1}$

- Hence, total number of nodes in a nearly complete binary tree, **$n= (2^{h+1} – 1) – (2^{h-1})$**

  (Writing 2^{h+1} in terms of 2^{h-1})

  $n = (4 * 2^{h-1} – 1) - (2^{h-1})$

  $= 3 * 2^{h-1} – 1$

  $2^{h-1} = (n + 1)/3$ --------------------(1)

- Maximum number of nodes in the left subtree of a nearly complete binary tree of **height h**,

  = $2^h - 1$ **(writing in terms of 2^{h-1})**

  $= 2 * 2^{h-1} - 1$ **---------------------(2)**

Substitute (1) in (2)

$= 2\,(\,(n + 1)/3\,) - 1$

$= (\,2n - 1\,)/\,3$

$<= 2n/3$

Hence, the worst case size of the subtree rooted at i, is at most 2n/3

- Children's subtrees each have size at most 2n/3

- The Running time of Max-Heapify ,
  $T(n) <= T(2n/3) + \Theta(1)$

- Using Master's Theorem (Case-2), the solution to this recurrence is , $T(n) = \Theta(\log n)$

- The **running time of Max-Heapify** on a node of **height h as O(h)**