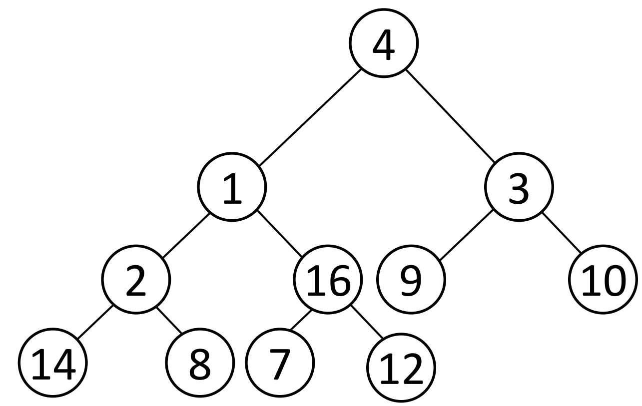
CS60020: Foundations of Algorithm Design and Machine Learning

Sourangshu Bhattacharya

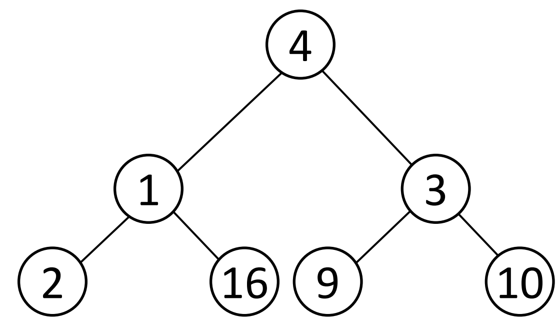
Special Types of Trees

- *Def:* Full binary tree = a binary tree in which each node is either a leaf or has degree exactly 2.



Full binary tree

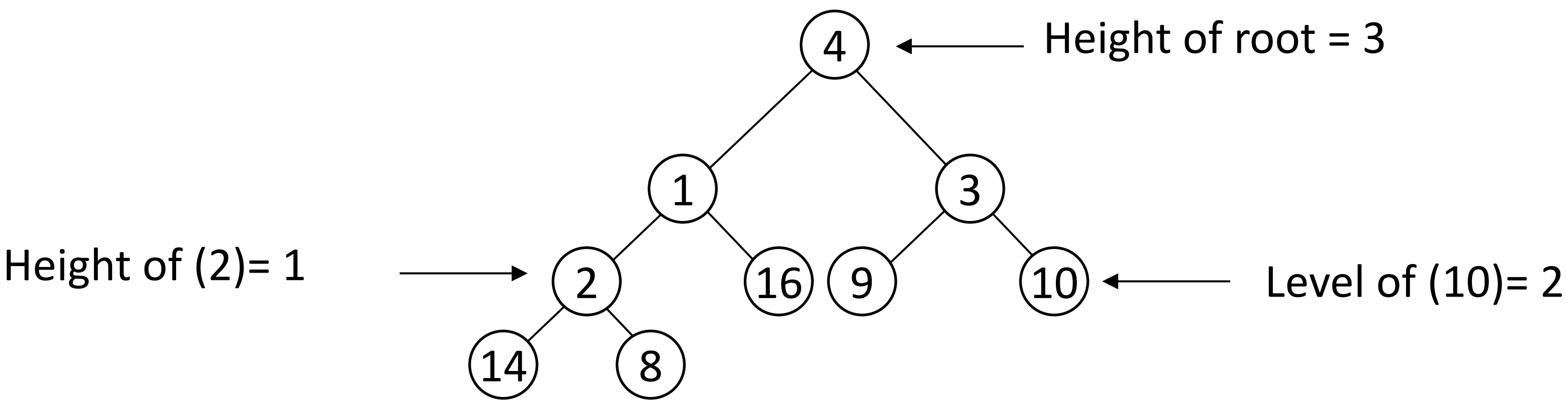
- *Def:* Complete binary tree = a binary tree in which all leaves are on the same level and all internal nodes have degree 2.



Complete binary tree

Definitions

- **Height** of a node = the number of edges on the longest simple path from the node down to a leaf
- **Level** of a node = the length of a path from the root to the node
- **Height** of tree = height of root node



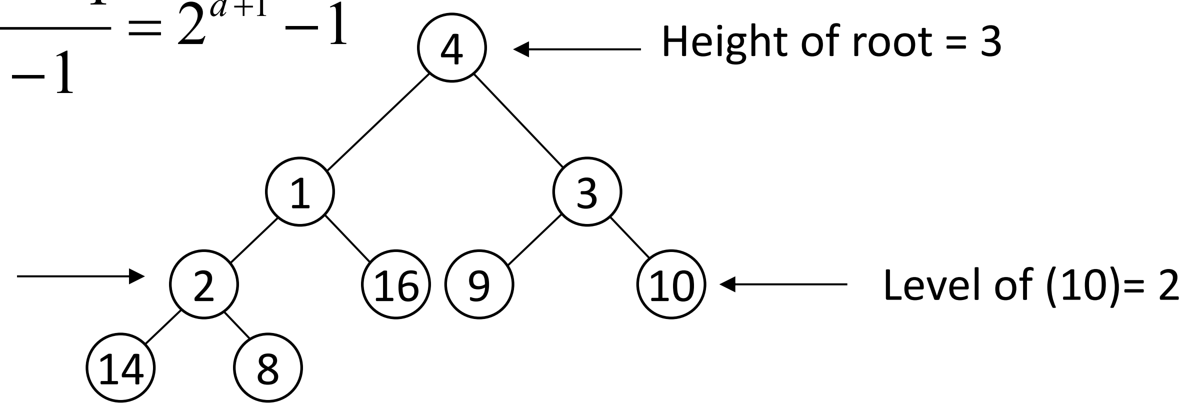
Useful Properties

- There are **at most** 2^l nodes at level (or depth) l of a binary tree
- A binary tree with height d has **at most** $2^{d+1} - 1$ nodes
- A binary tree with n nodes has height **at least** $\lfloor lgn \rfloor$

(see Ex 6.1-2, page 129)

$$n \le \sum_{l=0}^{d} 2^l = \frac{2^{d+1}-1}{2-1} = 2^{d+1}-1$$

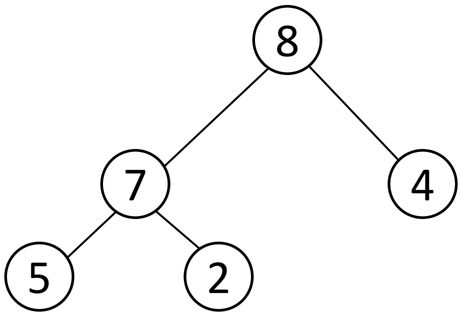
Height of (2)= 1



The Heap Data Structure

- *Def:* A **heap** is a nearly complete binary tree with the following two properties:
  - **Structural property:** all levels are full, except possibly the last one, which is filled from left to right
  - **Order (heap) property:** for any node x

$$\text{Parent}(x) \geq x$$



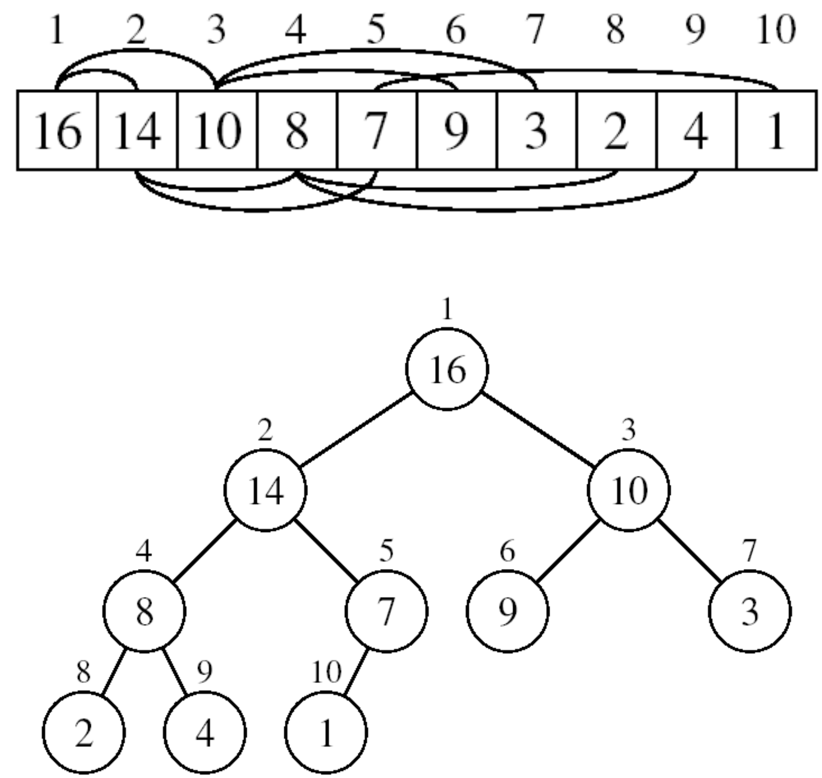
Heap

From the heap property, it follows that:
"The root is the maximum element of the heap!"

A heap is a binary tree that is filled in order

Array Representation of Heaps

- A heap can be stored as an array *A*.
  - Root of tree is $A[1]$
  - Left child of $A[i] = A[2i]$
  - Right child of $A[i] = A[2i + 1]$
  - Parent of $A[i] = A[\lfloor i/2 \rfloor]$
  - Heapsize[A] ≤ length[A]
- The elements in the subarray $A[(\lfloor n/2 \rfloor+1) .. n]$ are leaves



Heap Types

- **Max-heaps** (largest element at root), have the *max-heap property:*
  - for all nodes i, excluding the root:

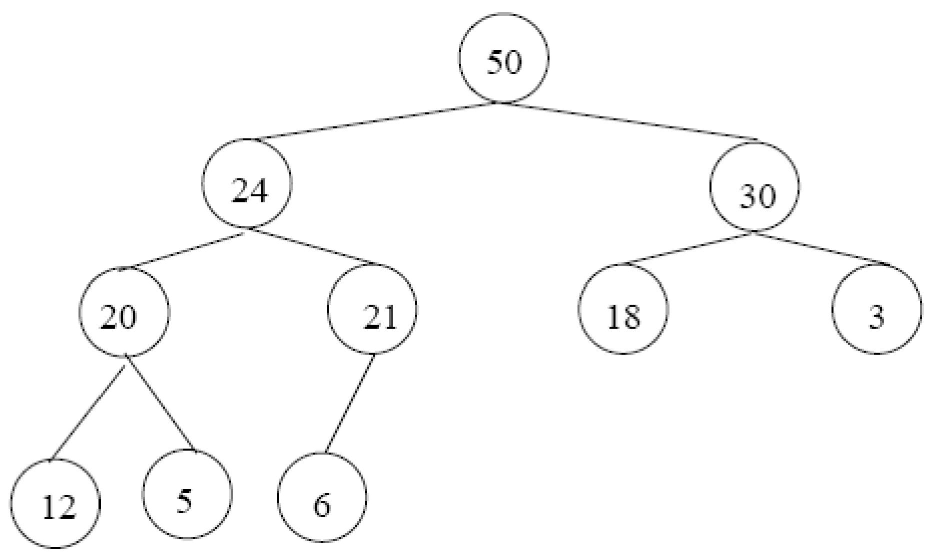
    $A[\text{PARENT}(i)] \geq A[i]$

- **Min-heaps** (smallest element at root), have the *min-heap property:*
  - for all nodes i, excluding the root:

    $A[\text{PARENT}(i)] \leq A[i]$

Adding/Deleting Nodes

- New nodes are always inserted at the bottom level (left to right)
- Nodes are removed from the bottom level (right

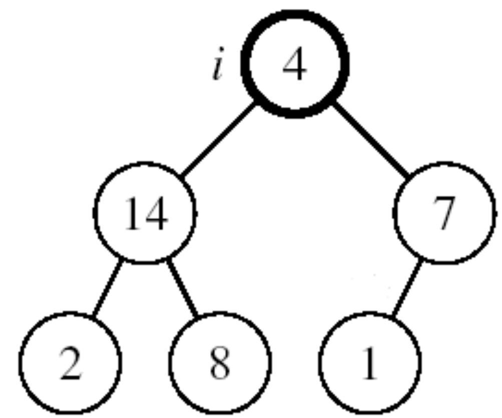


Operations on Heaps

- Maintain/Restore the max-heap property
  - MAX-HEAPIFY
- Create a max-heap from an unordered array
  - BUILD-MAX-HEAP
- Sort an array in place
  - HEAPSORT
- Priority queues

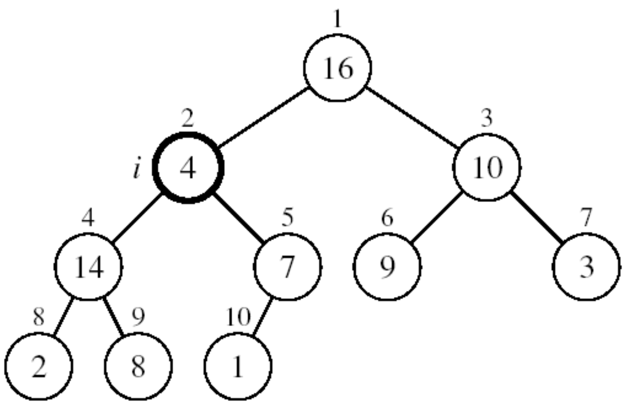
Maintaining the Heap Property

- Suppose a node is smaller than a child
  - Left and Right subtrees of i are max-heaps
- To eliminate the violation:
  - Exchange with larger child
  - Move down the tree
  - Continue until node is not smaller than children



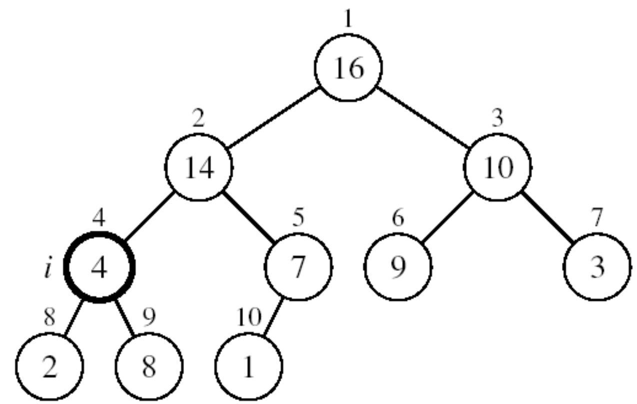
Example

<u>MAX-HEAPIFY(A, 2, 10)</u>



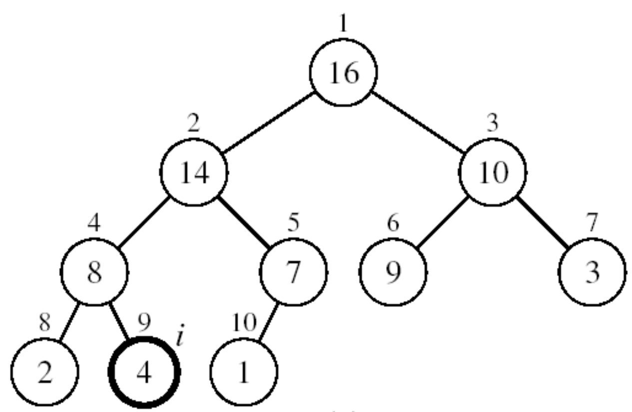
A[2] violates the heap property

A[2] ↔ A[4]



A[4] violates the heap property

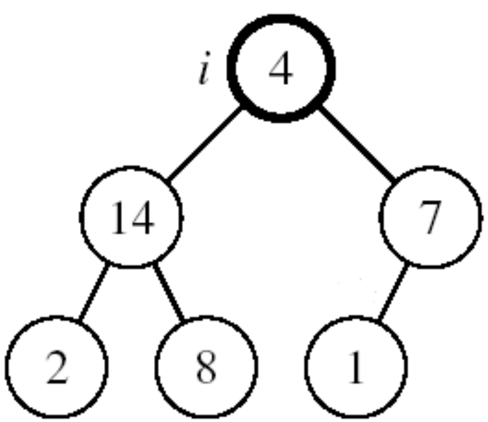
A[4] ↔ A[9]



Heap property restored

Maintaining the Heap Property

- Assumptions:
  - Left and Right subtrees of *i* are max-heaps
  - *A*[i] may be smaller than



Alg: MAX-HEAPIFY(*A*, i, n)

```
l ← LEFT(i)
r ← RIGHT(i)
if l ≤ n and A[l] > A[i]
    then largest ← l
    else largest ← i
if r ≤ n and A[r] > A[largest]
    then largest ← r
if largest ≠ i
    then exchange A[i] ↔ A[largest]
        MAX-HEAPIFY(A, largest, n)
```

MAX-HEAPIFY Running Time

- Intuitively:
  - It traces a path from the root to a leaf (longest path length: h)
  - At each level, it makes exactly 2 comparisons
  - Total number of comparisons is 2h
  - Running time is O(h) or $O(lgn)$

- Running time of MAX-HEAPIFY is $O(lgn)$

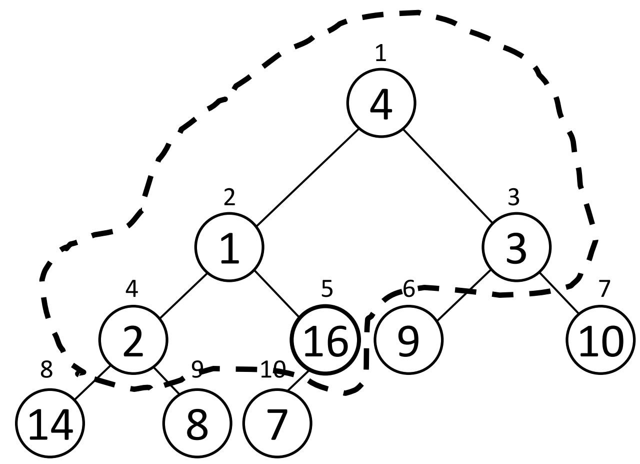
- Can be written in terms of the height of the heap, as being $O(h)$
  - Since the height of the heap is $\lfloor lgn \rfloor$

Building a Heap

- Convert an array $A[1 \ldots n]$ into a max-heap (n = $length[A]$)
- The elements in the subarray $A[(\lfloor n/2 \rfloor+1) \ .. \ n]$ are leaves
- Apply MAX-HEAPIFY on elements between 1 and $\lfloor n/2 \rfloor$

Alg: BUILD-MAX-HEAP(A)

1. n = length[A]
2. **for** i ← $\lfloor n/2 \rfloor$ **downto** 1
3. **do** MAX-HEAPIFY(A, i, n)



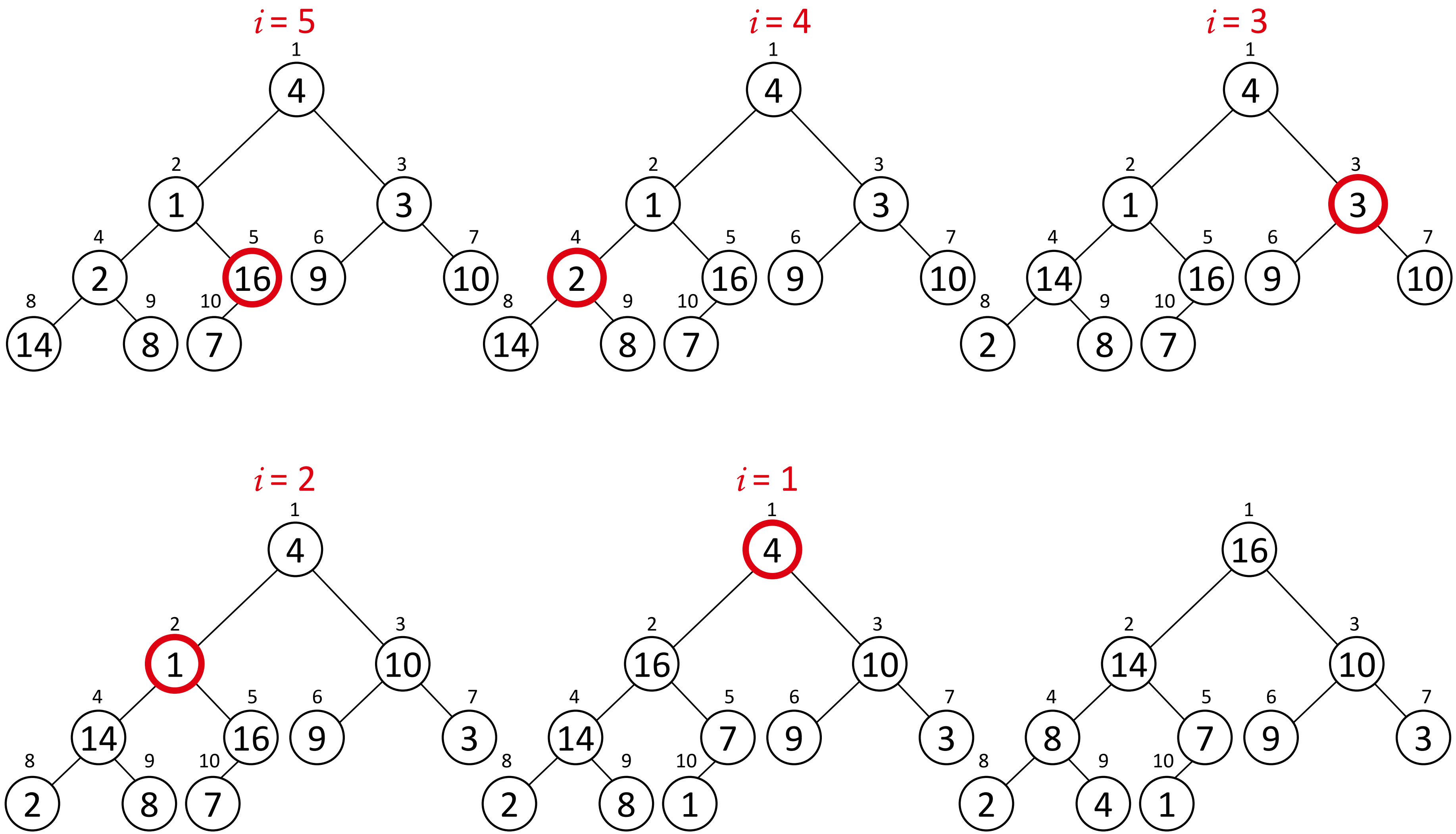
A:

4	1	3	2	16	9	10	14	8	7

Example:

A

4	1	3	2	16	9	10	14	8	7



Running Time of BUILD MAX HEAP

Alg: BUILD-MAX-HEAP(*A*)

1. n = length[A]
2. **for** i ← ⌊n/2⌋ **downto** 1
3. **do** MAX-HEAPIFY(*A*, i, n) — $O(\lg n)$ } $O(n)$

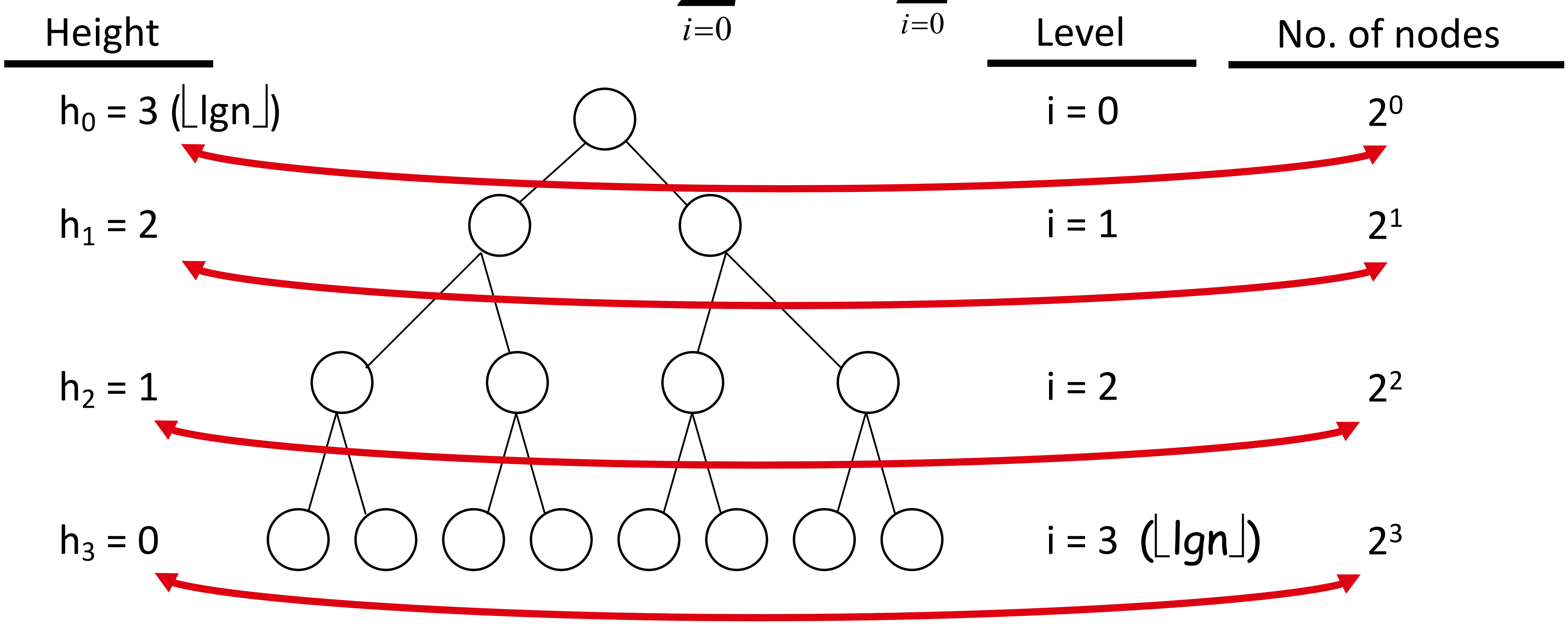
⇒ Running time: $O(n\lg n)$

- This is not an asymptotically tight upper bound

Running Time of BUILD MAX HEAP

- HEAPIFY takes $O(h)$ $\Rightarrow$ the cost of HEAPIFY on a node **i** is proportional to the height of the node **i** in the tree

$$\Rightarrow T(n) = \sum_{i=0}^{h} n_i h_i = \sum_{i=0}^{h} 2^i (h-i) = O(n)$$



$h_i = h - i$ height of the heap rooted at level i

$n_i = 2^i$ number of nodes at level i

Running Time of BUILD MAX HEAP

$$T(n) = \sum_{i=0}^{h} n_i h_i$$

Cost of HEAPIFY at level i ∗ number of nodes at that level

$$= \sum_{i=0}^{h} 2^i (h - i)$$

Replace the values of n_i and h_i computed before

$$= \sum_{i=0}^{h} \frac{h-i}{2^{h-i}} 2^h$$

Multiply by 2^h both at the nominator and denominator and write 2^i as $\frac{1}{2^{-i}}$

$$= 2^h \sum_{k=0}^{h} \frac{k}{2^k}$$

Change variables: k = h - i

$$\le n \sum_{k=0}^{\infty} \frac{k}{2^k}$$

The sum above is smaller than the sum of all elements to ∞ and h = lgn

$$= O(n)$$

The sum above is smaller than 2

Running time of BUILD-MAX-HEAP: T(n) = O(n)