Heaps

cse2011

section 8.3 of textbook

Priority Queues

- Data structure supporting the following operations:
  - ***insert*** (equivalent to enqueue)
  - ***deleteMin*** (or *deleteMax*) (equivalent to dequeue)
  - Other operations (optional)

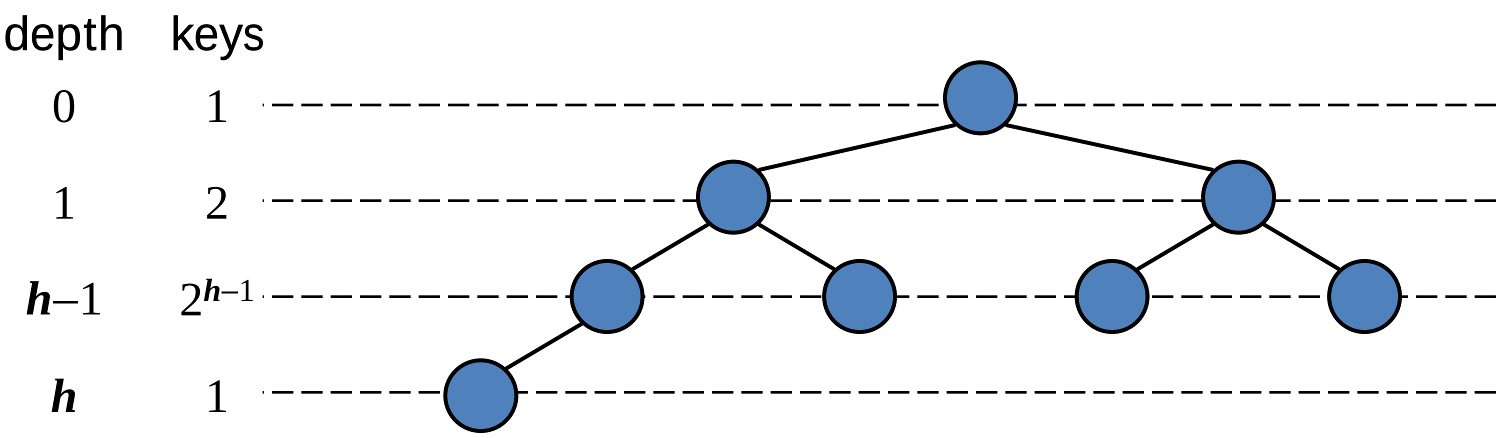
- Applications:
  - Emergency room waiting list
  - Routing priority at routers in a network
  - Printing job scheduling

Simple Implementations of PQs

- Unsorted linked list
  - insertion O(?)
  - deleteMin O(?)
- Sorted linked list
  - insertion O(?)
  - deleteMin O(?)
- Unsorted array
  - insertion O(?)
  - deleteMin O(?)
- Sorted array
  - insertion O(?)
  - deleteMin O(?)
- A data structure more **efficient** for PQs is heaps.

Complete Binary Trees

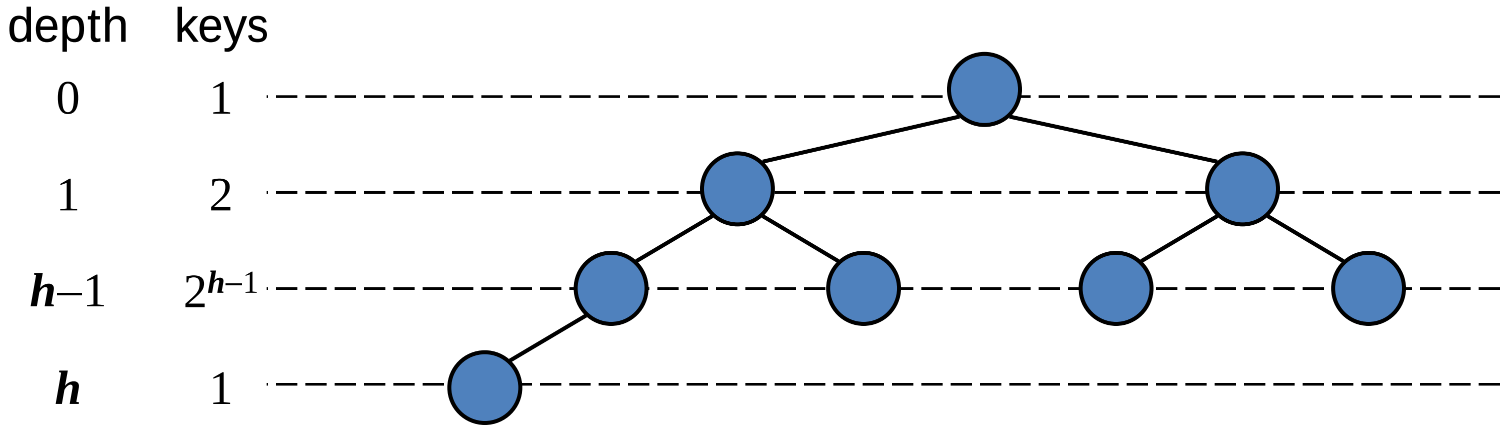
- Let $\boldsymbol{h}$ be the height of a binary tree.
  - for $\boldsymbol{i} = 0, \ldots, \boldsymbol{h} - 1$, there are $2^{\boldsymbol{i}}$ nodes at depth $\boldsymbol{i}$.
    - that is, all levels except the last are full.
  - at depth $\boldsymbol{h}$, the nodes are **filled from left to right**.



Complete Binary Trees (2)

- Given a complete binary tree of height *h* and size *n*, $2^{\boldsymbol{h}} \leq \boldsymbol{n} \leq 2^{\boldsymbol{h}+1} - 1$

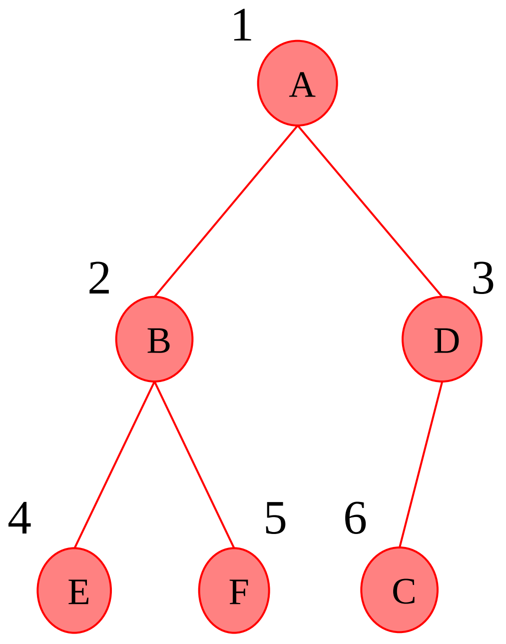
- Which data structure is better for **implementing** complete binary trees, arrays **or** linked structures?



Array Implementation of Binary Trees

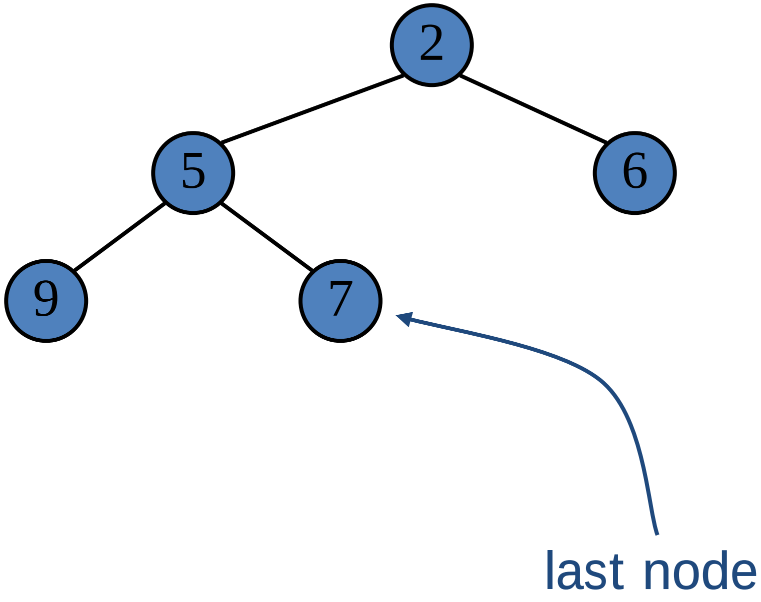
Each node ***v* is stored at index *i*** defined as follows:

- If *v* is the root, *i* = 1
- The **left** child of *v* is in position **2*i***
- The **right** child of *v* is in position **2*i* + 1**



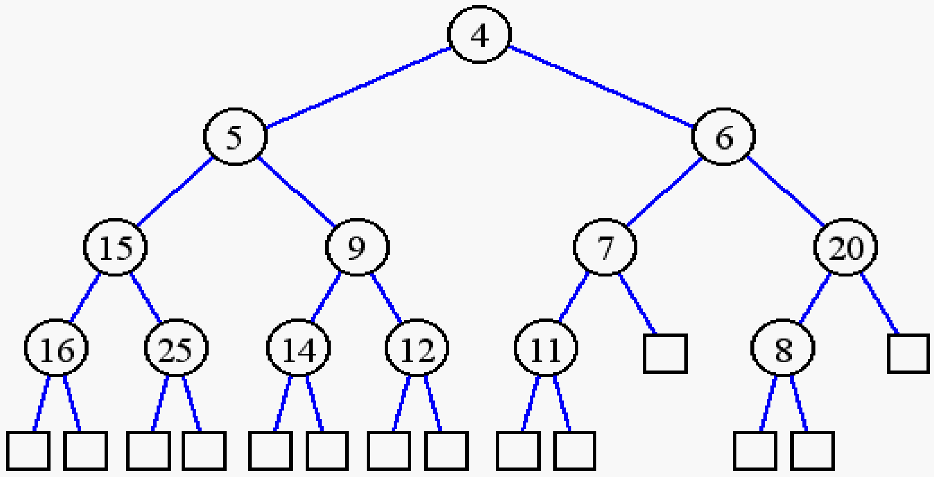
Heaps

- A heap is a **binary tree** storing keys at its nodes and satisfying the following properties:
  - Heap-Order: for every internal node v other than the root, $key(v) \geq key(parent(v))$
  - Complete Binary Tree: let h be the height of the heap
    - for $i = 0, \ldots, h - 1$, there are 2^i nodes at depth i.
    - at depth h, the nodes are filled from left to right.
- The **last node** of a heap is the **rightmost node** of depth h.
- Where can we find the **smallest** key in a min heap? The largest key?

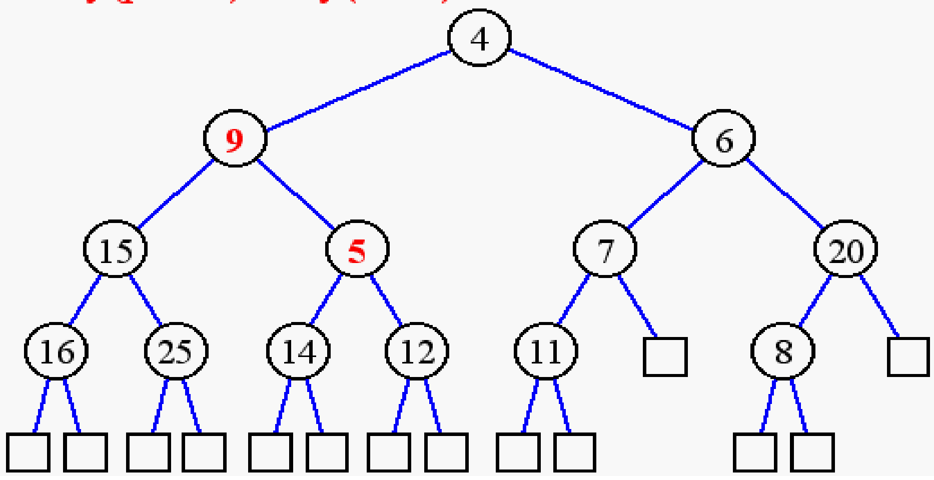


Examples that are not heaps

- bottom level is not left-filled



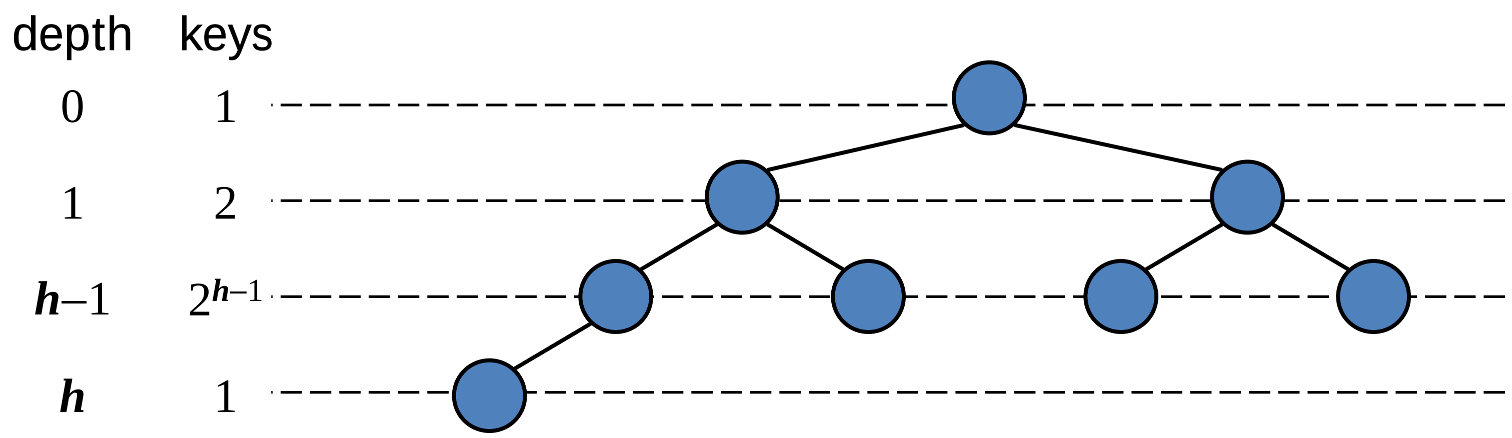
- key(parent)> key(child)



Height of a Heap

- Theorem: A heap storing n keys has height $O(\log n)$

  Proof: (we apply the complete binary tree property)
  - Let h be the height of a heap storing n keys
  - Since there are 2^i **keys** at **depth** $i = 0, \ldots, h-1$ and **at least one key at depth** h, we have $n \geq 1 + 2 + 4 + \ldots + 2^{h-1} + 1$
  - Thus, $n \geq 2^h$, i.e., $h \leq \log n$

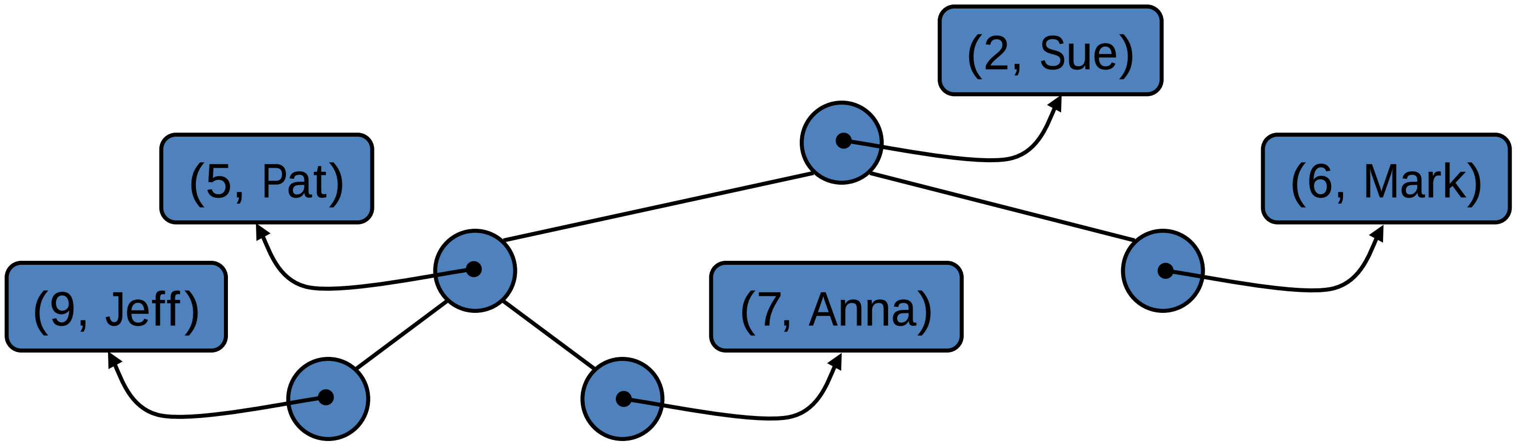


Max Heap

- The definition we just discussed is for a ***min* heap**.
- Analogously, we can declare a ***max* heap** if we need to implement *deleteMax* operation instead of *deleteMin*.

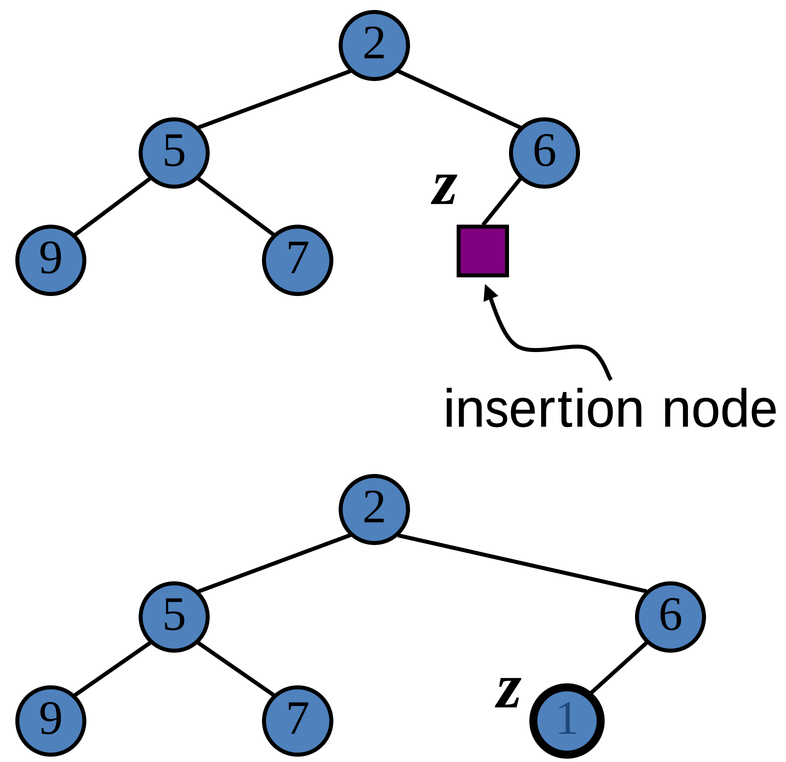
Heaps and Priority Queues

- We can use a heap to implement a priority queue
- We store a **(key, element)** item at each internal node
- We keep track of the position of the last node



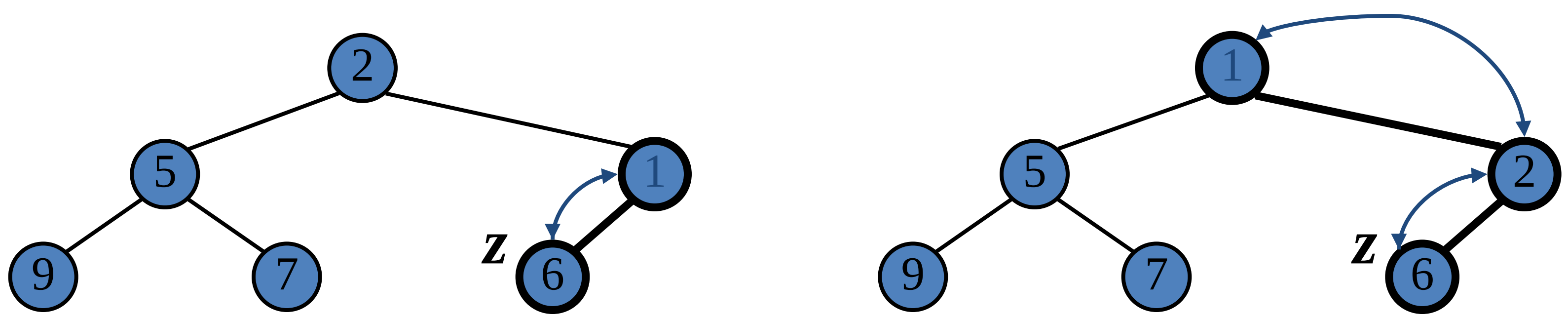
Insertion into a Heap

- Method insert of the priority queue ADT corresponds to the insertion of a key k to the heap
- The insertion algorithm consists of **three steps**
  – Find the insertion node z (the new last node)
  – Store k at z
  – Restore the heap-order property (discussed next)



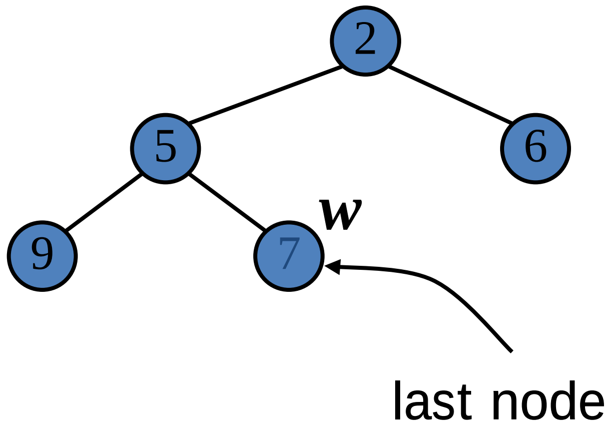
Up-heap Percolation (up-heap bubbling)

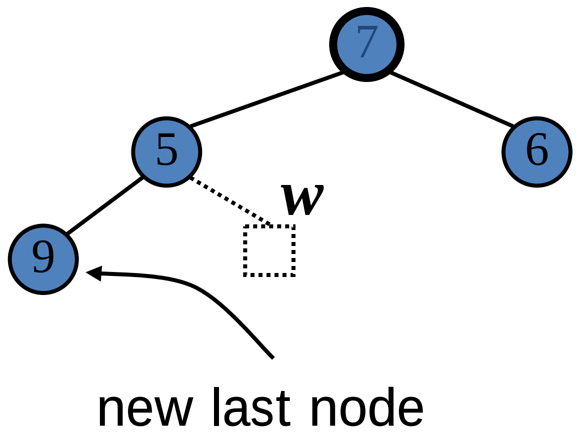
- After the insertion of a new key $\boldsymbol{k}$, the heap-order property may be violated
- Algorithm upheap restores the heap-order property by swapping $\boldsymbol{k}$ along an upward path from the insertion node
- Upheap terminates when the key $\boldsymbol{k}$ reaches the root or a node whose parent has a key smaller than or equal to $\boldsymbol{k}$
- Since a heap has height $\boldsymbol{O}(\log \boldsymbol{n})$, upheap runs in $\boldsymbol{O}(\log \boldsymbol{n})$ time



Removal from a Heap

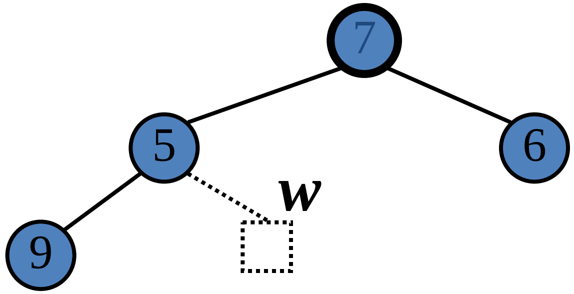
- Method *deleteMin* of the priority queue ADT corresponds to the removal of the root key from the heap
- The removal algorithm consists of **three steps**
  - Replace the root key with the key of the last node ***w***
  - Remove ***w***
  - Restore the heap-order property (discussed next)

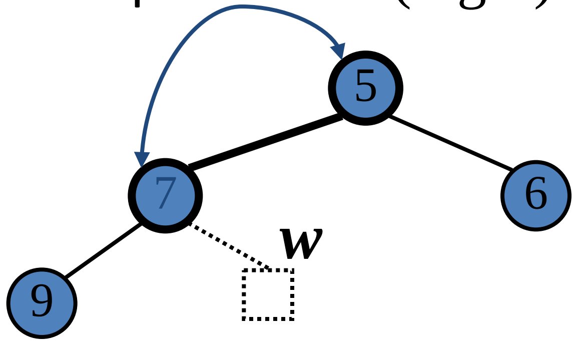




Down-heap Percolation
Down-heap bubbling

- After replacing the root key with the key $\boldsymbol{k}$ of the last node, the heap-order property may be violated
- Algorithm down-heap restores the heap-order property by swapping key $\boldsymbol{k}$ along a downward path from the root
- If the node has two children, it is swapped with the one which has the smallest key.
- Up-heap terminates when key $\boldsymbol{k}$ reaches a leaf or a node whose children have keys greater than or equal to $\boldsymbol{k}$
- Since a heap has height $\boldsymbol{O}(\log \boldsymbol{n})$, down-heap runs in $\boldsymbol{O}(\log \boldsymbol{n})$ time





Next lecture …

- Heap Sort
- Hash Tables, part 1