1 Simple Implementations of a Priority Queue

- Implementation with an unsorted list
  - `insert(n)` is $O(1)$
  - `min(n)` and `removeMin(n)` is $O(n)$
- Implementation with an sorted list
  - `insert(n)` is $O(n)$
  - `min(n)` and `removeMin(n)` is $O(1)$ (always at head)

2 Binary heap

A heap is a binary tree storing key-value pairs 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 $i = 0, \ , h-1$, there are 2^i nodes of depth i
  - At depth $h-1$, the nodes are to the **left** of any **missing nodes**

2.1 Height

Theorem: A heap storing n keys has height $O(log\mathbf{n})$ Proof: This uses just the complete binary tree property

2.2 Insertion into a Heap

Method `insertItem` 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)

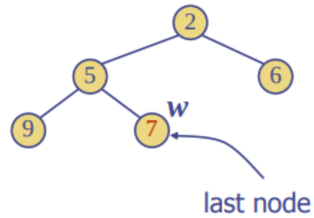
2.3 Unheap

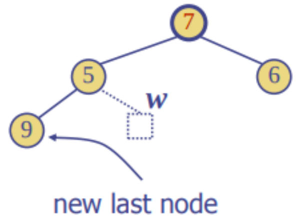
- After the insertion of a new key k, the heap-order property may be violated
- Algorithm upheap restores the heap-order property by swapping k along an **upward path** from the insertion node
- Upheap terminates when the key k reaches the root or a node whose parent has a key **smaller than or equal**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

2.4 Removal from a Heap

Method `removeMin` 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)





2.5 Downheap

After replacing the **root** key with the key k of the **last** node, the *heap-order property* may be violated. Algorithm *downheap* restores the heap-order property by **swapping** key k along a particular downward path from the root. Downheap terminates when key k reaches a leaf or a node whose children have keys **greater than or equal** to k. Since a heap has height $O(logn)$, downheap runs in $O(logn)$ time

2.6 Array List Heap Implementation

- We can represent a heap with n keys by means of a vector or ArrayList of length n + 1
- Links between nodes are not explicitly stored, instead:
- For the node at index i:
  - Left child is at index $2i$
  - Right child is at index $2i + 1$
  - Parent is at index $i/2$
- The cell of at index 0 is not used (Would mess up children indixes)
- Notice that there are **no gaps** when storing a heap
- Operation `insert` corresponds to inserting at index $n + 1$
- Operation `removeMin` corresponds to moving index n to index 1
- Up- and down-heap operations just swap appropriate elements within the array
- Together with the **lack of gaps**, this makes the implementation very **efficient**, and this is the standard way to implement a Heap.