Heaps Efficiently Implement Priority Queues

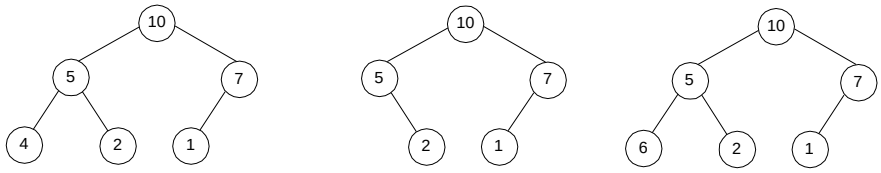
- Heap is an efficient way of supporting an algorithm that requires repeatedly finding the maximum (or minimum) of a data set.
- In particular, heaps support
  - Remove max – O(log n) time
  - Insert – O(log n) time
  - Build heap – O(n) time
- Examples of problems where Heaps are useful
  - **Priority Queues**
  - Simulations
  - Sorting

Max Heaps (almost identical for Min Heaps)

- Definition A heap is a binary tree with keys at its nodes (one key per node) such that:
- It is *essentially complete*,
  - i.e., all tree levels are full except possibly the last level, where only rightmost keys may be missing – **shape property**.

- The key at each node is ≥ keys at its children (MaxHeap)
  - **heap property** (sometimes called parental dominance)

Illustration of the heap's definition



a heap **not a heap** **not a heap**

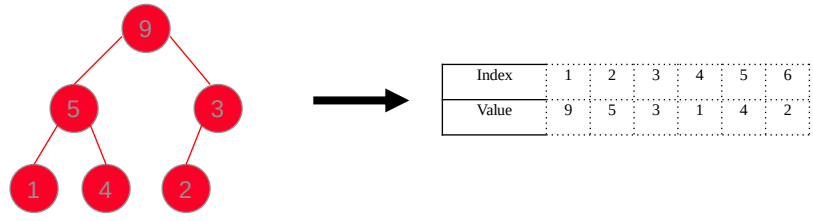
Note: Heap's elements are ordered along any path from its root to leaf, but they are not ordered left to right

Some Important Properties of a Heap (MaxHeap)

- Given *n,* there exists a unique binary tree with *n* nodes that is essentially complete, with $h = \lfloor \log_2 n \rfloor$
- The root contains the largest key
- The subtree rooted at any node of a heap is also a heap
- A heap can be efficiently implemented using an array

(Binary) Heap's Array Representation

- Store heap's elements in an array (whose elements indexed, for efficiency, 1 to n) in top-down left-to-right order
- Example:



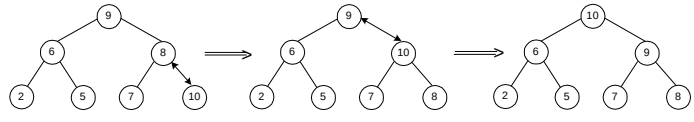
- Left child of node j is at 2j Right child of node j is at 2j+1
- Parent of node j is at ⌊j/2⌋
- Parental nodes are represented in the first ⌊n/2⌋ locations

Insertion of an element into a heap

- Insert the new element after the position of the last element in heap. This maintains structure property
- Compare it with its parent and, if it violates heap condition, exchange them (Drift up)
- Continue comparing the new element with nodes up the tree until the heap condition is satisfied

Example: Insert key 10

Efficiency: O(log *n*) since the height of the tree is log n



Insertion into heap: (Drift up or Percolate up)

- Use first open position in the array to make room for new element.
- Drift the hole up until find place for new element that maintains the heap property

```
A is array
Insert( Item x)
    check heap capacity, increase if necessary
    increase heap size by 1 make space for additional item
    int hole = heapSize
    while ((hole>1) && x>heap[hole/2]))
       A[hole]=A[hole/2]    // move parent down
       hole = hole/2        // move open position up
    a[hole]=x               // put x where it belongs
```


Return and remove max from heap

```
Store first entry (max) for return
Move the last entry in heap to first entry in heap
Reduce heapsize by 1
Drift-down the new first element until the heap property is
       restored
Return the original root of the heap
```

Note: There are only two basic "moves" in the heap.

- *Drift-down* – used in removal and bottom up construction
- *Drift-up* – used in insertion and top-down construction

Remove max uses: Drift down or Percolate down

```
a is array containing the heap
driftDown( int pos)
    tmp=array[pos]  // save value want to drift down
    while(pos * 2 <= heapSize)  // pos is parent node
        child = pos * 2
        if( child!=heapSize && (a[child + 1]>a[child]))
            child++
        if array[ child ] > tmp  )  //array[child] bigger
            a [ pos ] = a [ child ] // move it up
            pos = child             // drift down pos
        else
            break
    a [pos] = tmp
```


A Priority Queue is an ADT

- Useful in many contexts: process scheduling, shortest paths,…
- Many implementation options : list, sorted list, …
- With following operations:
  - find element with highest priority
  - delete element with highest priority
  - insert element with assigned priority
- Enhance with
  - Delete a given element
  - Change key for a given element
  - The essential idea is to be able to find the element in the heap in constant time! Maintain an additional array for the objects in the heap that contains their position in the (heap) priority queue (handle)