Binary Heaps

- **Complete** Binary Tree
- Maintains the **Heap** property

- Reading:
  – ODS Section 10.1
PriorityQueue Interface

- **add(x)** – add item x to the Queue

- **remove() / deleteMin()** – remove the smallest item from the queue

- **peek() / findMin()** – get the smallest item without deleting it

  – Note: with a different comparator, the queue could find largest, etc....
Binary Heaps

Theorem:

A **BinaryHeap** implements the **(priority) Queue** interface. Ignoring the cost of resizing() operations, a BinaryHeap supports the operations:

- **add(x)/remove()** in **$O(\log n)$** time per operation.
- **peek()** in **$O(1)$** time per operation.

In addition, starting with an empty BinaryHeap, any sequence of $m$ add(x) and remove() operations results in a total of **$O(m)$** time spent during all calls to **resize()**.
HeapSort

- Sorting algorithm that uses a heap
- **In-place** sorting
- Comparison-based
- Runs in $O(n \log n)$ time

Reading:
- ODS Section 11.1.3
HeapSort

Theorem:

The **HeapSort** algorithm runs in $O(n \log n)$ time and performs at most $2n \log n + O(n)$ comparisons.
Meldable Heaps

- Randomized heap
- Supports the \texttt{meld(h1,h2)} operation
- Works w/ non-complete binary trees

\begin{center}
\begin{tikzpicture}
  \node[shape=circle,draw=blue] (2) at (1,4) {2};
  \node[shape=circle,draw=blue] (7) at (1,2) {7};
  \node[shape=circle,draw=blue] (10) at (2,2) {10};
  \node[shape=circle,draw=blue] (8) at (2,0) {8};
  \node[shape=circle,draw=blue] (3) at (4,4) {3};
  \node[shape=circle,draw=blue] (6) at (4,2) {6};
  \node[shape=circle,draw=blue] (4) at (5,2) {4};
  \node[shape=circle,draw=blue] (7) at (9,6) {7};
  \node[shape=circle,draw=blue] (3) at (9,3) {3};
  \node[shape=circle,draw=blue] (10) at (10,3) {10};
  \node[shape=circle,draw=blue] (8) at (10,0) {8};
  \node[shape=circle,draw=blue] (6) at (11,0) {6};
  \node[shape=circle,draw=blue] (4) at (12,0) {4};
  \path[->,blue]
  (2) edge (7)
  (2) edge (10)
  (3) edge (6)
  (3) edge (4)
  (7) edge (8)
  (7) edge (6)
  (3) edge (8)
  (3) edge (6)
  (7) edge (10)
  (7) edge (6)
  (7) edge (4)
  (3) edge (8)
  (3) edge (6)
  (3) edge (4);
\end{tikzpicture}
\end{center}

- Reading:
  - ODS Section 10.2
Random Walks

• A Random walk is a path through a binary tree
  – Starting at the root of the tree
  – At each step toss a coin to decide left or right child
  – End when walking off of the tree

Theorem: The expected length of a random walk in a binary tree of size $n$ $(E[W_n])$ is $< \log(n+1)$
Meldable Heaps

Theorem:

A MeldableHeap implements the (priority) Queue interface. A MeldableHeap supports the operations:

- `add(x)/remove()` in $O(\log n)$ expected time per operation.
- `peek()` in $O(1)$ time per operation.