⇒ Min Priority Heap
Min Priority Heap (Chapter 6)

- **Binary Search Tree**
  - Insert Search Min in $O(\log(n))$ (average case)
- **What if**
  - You do not need search?
  - You want guaranteed $O(\log(n))$ time?
  - Want lower amount of overhead (smaller constant)?
- **Use a min priority heap**
Basic Idea

- It can be min or max priority heap (using max here)
- Ensure node value greater than everything in subtree below it
  - Less restrictive than a binary search tree
  - Makes it possible to keep the tree very bushy than a binary search tree

- Bushy tree:
  - Each layer filled left to right before next layer is started
  - Use array to store complete tree
Accessing Parent and Children

- Textbook starts array at 1 (homework will start at 0)
- How do we access
  - A node’s parent?
  - Left child?
  - Right child?
  - Max element?
- Maximum height of tree with n nodes?
Getting the Array into Max Heap format

• Start with an array that is in some random order
• Start at the last node \( n \), and work towards the first node
  - Make sure subtree rooted at \( n \) obeys max-heap property
    + If \( n \) is greater or equal to its maximal child, nothing to do
    + Otherwise swap \( n \) with its maximal child, and keep going down the tree
Example

```
1 14 3 2 16 8 4 10 7 9
...
1 14 3 2 16 8 4 10 7 9
1 14 3 10 16 8 4 2 7 9
1 14 3 10 16 8 4 2 7 9
1 14 3 10 16 8 4 2 7 9
1 14 3 10 16 8 4 2 7 9
1 16 8 10 14 3 4 2 7 9
1 16 8 10 14 3 4 2 7 9
1 16 8 10 14 3 4 2 7 9
1 16 8 10 14 3 4 2 7 9
1 16 8 10 14 3 4 2 7 9
1 16 8 10 14 3 4 2 7 9
```
**Code**

\[\text{MAX-HEAPIFY}(A, i)\]

1. \( l = \text{LEFT}(i) \)
2. \( r = \text{RIGHT}(i) \)
3. \textbf{if} \( l \leq A.\text{heap-size} \) \text{ and } \( A[l] > A[i] \) \textbf{then}
   \( \text{largest} = l \)
4. \textbf{else}
   \( \text{largest} = i \)
5. \textbf{if} \( r \leq A.\text{heap-size} \) \text{ and } \( A[r] > A[\text{largest}] \)
6. \( \text{largest} = r \)
7. \textbf{if} \( \text{largest} \neq i \)
8. \text{exchange} \( A[i] \) \text{ with } \( A[\text{largest}] \)
9. \( \text{MAX-HEAPIFY}(A, \text{largest}) \)
Build-Max-Heap

Build-Max-Heap(A)

1. \( A.heap-size = A.length \)
2. \textbf{for } i = \lfloor A.length/2 \rfloor \textbf{ downto } 1
3. \textbf{Max-Heapify} \((A, i)\)

- \( A.heap-size \) versus \( A.length \)
- Why that starting point?
- Running time?

© P. Heeman, 2017
Extracting Max

**Heap-Extract-Max(A)**

1. **if** $A$.$heap$-$size$ $<$ 1
2. **error** “heap underflow”
3. $max = A[1]$
5. $A$.heap$-$size$ = $A$.heap$-$size$ $-$ 1
6. **Max-Heapify** ($A$, 1)
7. **return** $max$
**Inserting**

**HEAP-INCREASE-KEY** \((A, i, key)\)

1. **if** \(key < A[i]\)
2. **error** "new key is smaller than current key"
3. \(A[i] = key\)
4. **while** \(i > 1\) and \(A[\text{PARENT}(i)] < A[i]\)
5. exchange \(A[i]\) with \(A[\text{PARENT}(i)]\)
6. \(i = \text{PARENT}(i)\)

• How can we use this to insert?