# Priority Queues & Heaps II

**Binary Heap - Extract Min**
- Min key at root \( \rightarrow \) save it
- Copy \( A[\text{ln}] \) to root (\( A[1] \))
- Decrement \( n \)
- Sift the root key down
  - Find smaller of two children
  - If larger, swap with this child
- Return saved root key

**Leftist Property:**
- Null path length
  \( npl(v) = \) length of shortest path to null

\[
\begin{align*}
\text{null} & \quad \text{if } v = \text{null} \\
\text{null} + \min \left( \text{npl}(v.\text{left}), \text{npl}(v.\text{right}) \right) & \quad \text{otherwise}
\end{align*}
\]

**Def:** Leftist Heap is a binary tree where:
- Keys are \( \text{heap ordered} \)
- All nodes \( v, \text{npl}(v.\text{left}) \geq \text{npl}(v.\text{right}) \)

**Examples:**

Leftist Heaps: Meldable heaps
- Can merge two heaps into single heap
- Eg. One processor breaks, awaiting jobs must be merged with another processor.

**Analysis:** Both insert \& extract-min take time proportional to tree height
- Tree is complete \( \Rightarrow \mathcal{O}(\log n) \) time
Class structure:

```
Class LeftistHeap
  private class LHNode
    public LeftistHeap()
      ... (constructor)
    void insert(Key x)
    void extractMin()
    void mergeWith(LeftistHeap H2)
  ... (other private/protected utilities)
```

Lemma: A leftist tree with \( r \geq 1 \) nodes along its rightmost path has \( n \geq 2^{(r-1)} \) nodes.

Proof: (Sketch - see later notes)

- By induction: \( n_1 = 2^{(r-1)}, n_2 \geq 2^{(r-1)} \)
- \( n = 1 + n_1 + n_2 \geq 2^{(r-1)} + 2^{(r-1)} = 2^{(r-1)} \)

Analysis: Time \( n \) Rightmost path

```
Insert + Extract-Min? Exercises
```

```
LHNode merge(LHNode u, LHNode v)
  if (u == null) return v
  if (v == null) return u
  if (u.key > v.key) // swap so u is smaller
    swap u,v
  if (u.left == null) u.left <- v
  else
    u.right <- merge (u.right, v)
    if (u.left.npl < u.right.npl)
      swap u.left <- u.right
    u.npl <- u.right.npl + 1
  return u
```

Merge helper: 2 phases

1. Merge right paths by order of keys + update npl's
2. Check leftist property + swap
Dictionary:
- **insert** (Key x, Value v)
  - insert (x,v) in dict. (No duplicates)
- **delete** (Key x)
  - delete x from dict. (Error if x not there)
- **find** (Key x)
  - returns a reference to associated value v, or null if not there.

Search: Given a set of n entries each associated with key x:
- and value v:
  - Store for quick access & updates
- Ordered: Assume that keys are totally ordered: <, >, ==

Binary Search Trees
- Basic definitions
- Finding keys

Can we achieve O(log n) time for all ops? **Binary Search Trees**

Idea: Store entries in binary tree sorted (inorder traversal) by key

Efficiency: Depends on tree's height
- Balanced: \(O(\log n)\)
- Unbalanced: \(O(n)\)

Example:
```
find(5)
```

```
A
Can we achieve O(log n) time for the tree?
- Start at root p=root
  - if (x < p.key) search left
  - if (x > p.key) search right
  - if (x == p.key) found it!
  - if (p == null) not there!
```
Announcements: Tue 2/14

- Homework 1
  - Due: Tue, Feb 21, start of class
    → No late submissions

- Programming Assignment 1
  - Leftist Heaps
  - Almost ready
  - Due, Wed, Mar 1

Project 1:
- Leftist Heap
  - Max heap
    - Weight-based
      - not npl
  - updateKey
    → Locators
    → Parent links

Locator loc = insert(Key x, Value v)