CMSC 341
Leftist Heaps

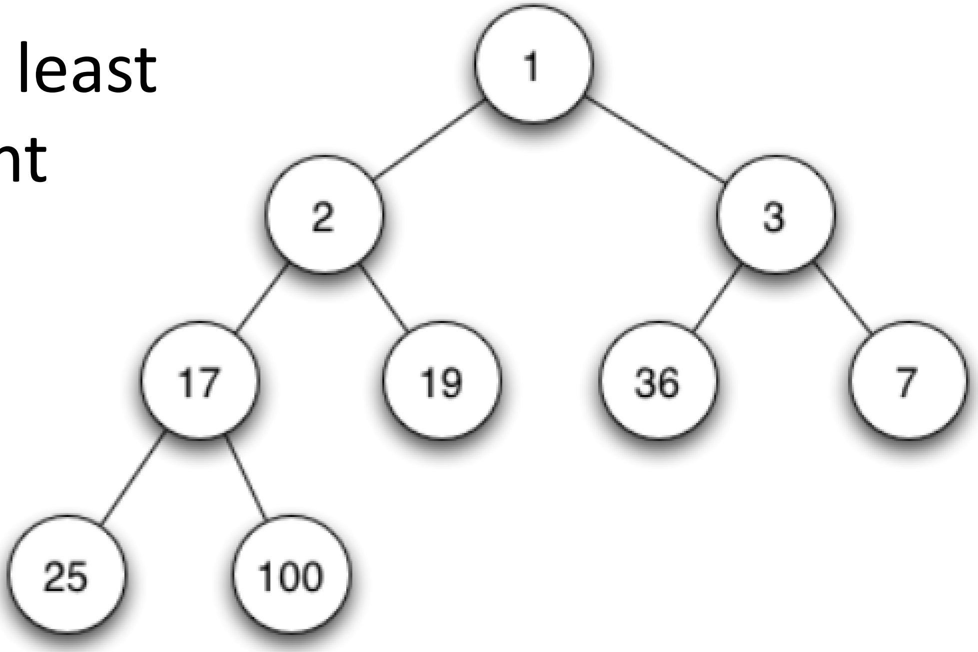
Today’s Topics

- Review of Min Heaps
- Introduction of Left-ist Heaps
- Merge Operation
- Heap Operations

Review of Heaps

Min Binary Heap

- A **min binary heap** is a…
  - Complete binary tree
  - Neither child is smaller than the value in the parent
  - Both children are at least as large as the parent
- In other words, smaller items go above larger ones



Min Binary Heap Performance

- Performance
  - (n is the number of elements in the heap)

- construction `O( n )`
- **`findMin()`** `O( 1 )`
- **`insert()`** `O( lg n )`
- **`deleteMin()`** `O( lg n )`

Introduction to Leftist Heaps

Leftist Heap Concepts

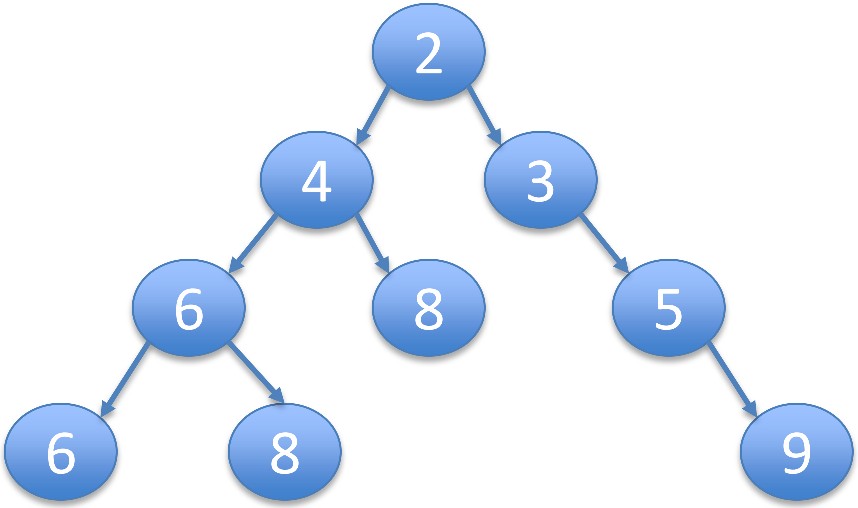
- Structurally, a leftist heap is a **min tree** where each node is marked with a **rank** value.
- Uses a Binary Tree (BT)!!
- Merging heaps is much easier and faster
  - May use already established links to merge with a new node
  - Rather than copying pieces of an array

Leftist Heap Concepts

- Values STILL obey a heap order (partial order)
- Uses a null path length to maintain the structure (covered in detail later)
  - **The null path of a node's** ***left child is >= null path of the right child***
- At every node, the shortest path to a non-full node is along the rightmost path!!!

Leftist Heap Example

- A leftist heap, then, is a purposefully **unbalanced** binary tree (leaning to the left, hence the name) that keeps its smallest value at the top and has an inexpensive merge operation



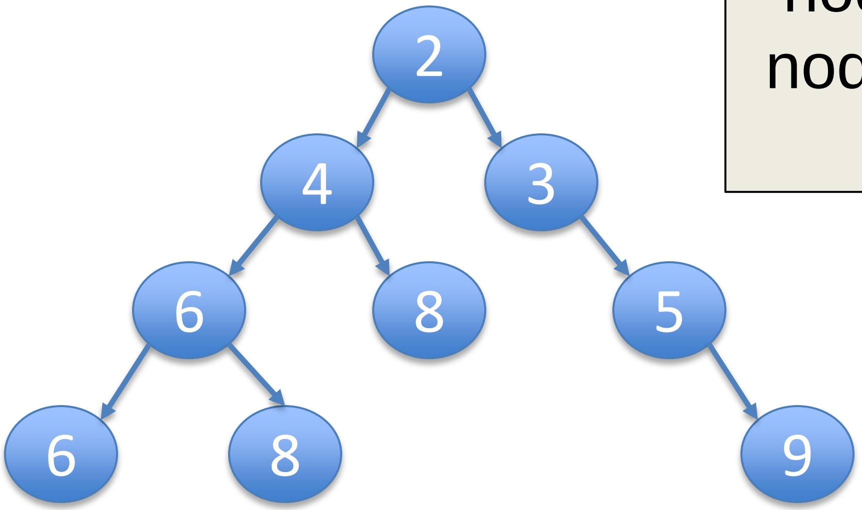
Leftist Heap Performance

- Leftist Heaps support:
  - `findMin()` = `O(1)`
  - `deleteMin()` = `O(log n)`
  - `insert()` = `O(log n)`
  - construct = `O(n)`
  - `merge()` = `O(log n)`

Null Path Length (npl)

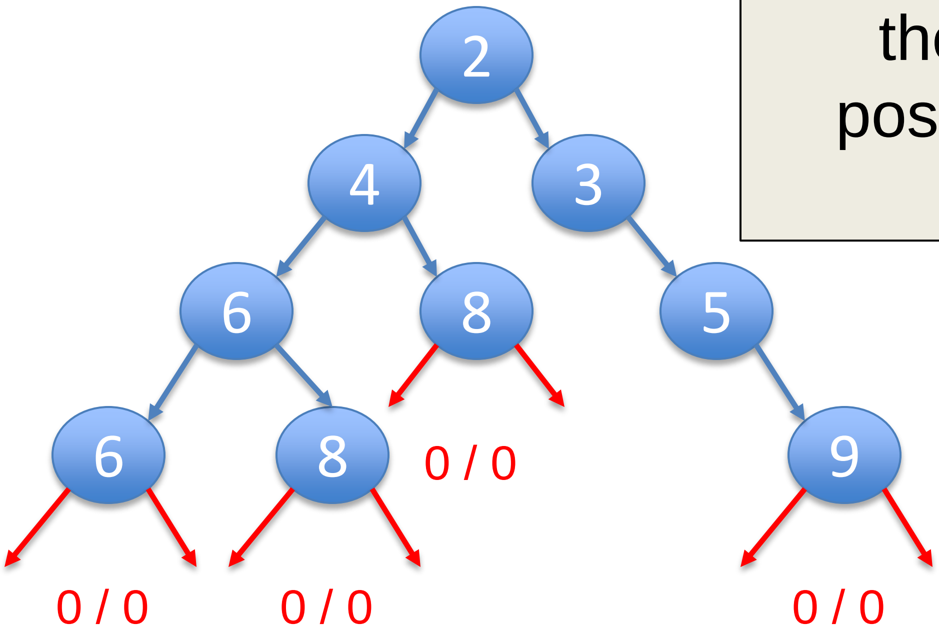
- Length of **shortest** path from current node (X) to a node **without** 2 children
  - value is stored IN the node itself
- leafs
  - npl = 0
- nodes with only 1 child
  - npl = 0

Null Path Length (npl) Calculation



To calculate the npl for each node, we look to see how many nodes we need to traverse to get to an open node

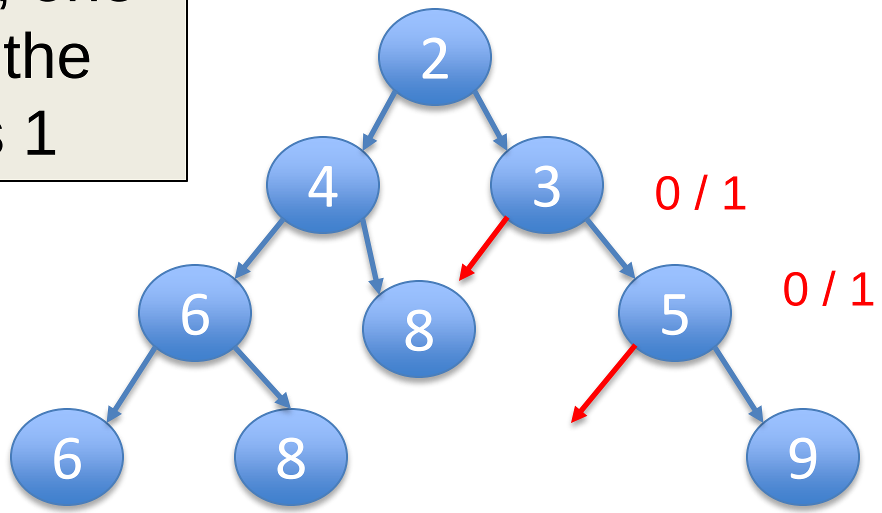
Null Path Length (npl) Calculation



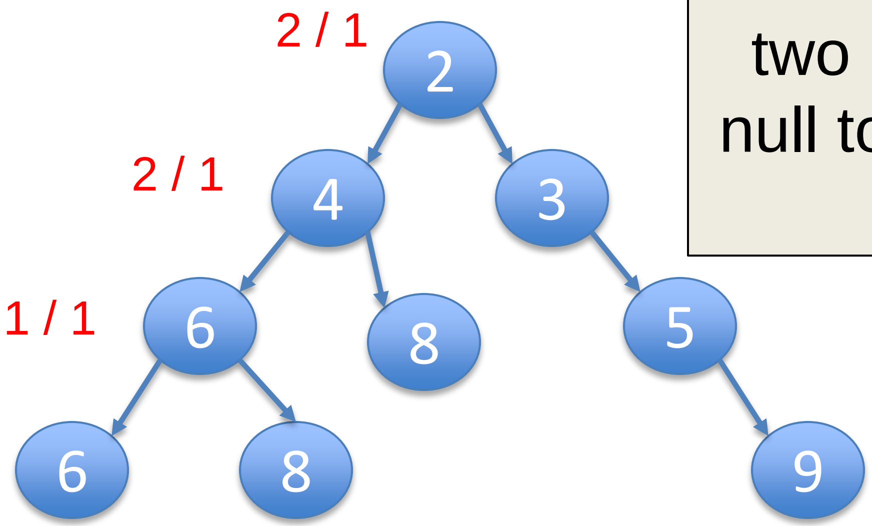
In the leaves case, there is a null position 0 nodes away

Null Path Length (npl) Calculation

In these cases, one side is 0 and the other side is 1



Null Path Length (npl) Calculation



In the root, it will take two levels to get to null to the left; one to the right.

Leftist Node

- The node for a leftist heap will have many member variables this time
  - links (left and right)
  - element (data)
  - npl
- By default, the Leftist Heap sets an empty node as the root

Leftist Node Code

```
private:
    struct LeftistNode
    {
        Comparable   element;
        LeftistNode *left;
        LeftistNode *right;
        int          npl;

        LeftistNode( const Comparable & theElement, LeftistNode *lt = NULL,
                        LeftistNode *rt = NULL, int np = 0 )
          : element( theElement ), left( lt ), right( rt ), npl( np ) { }
    };

    LeftistNode *root;
```

Looks like a binary tree node except the npl being stored.

Building a Leftist Heap

Building a Leftist Heap

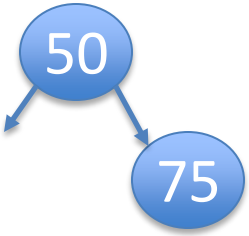
- Value of node STILL matters
  - Lowest value will be root, so still a min Heap
- Data entered is random

- Uses CURRENT npl of a node to determine where the next node will be placed

Moves in Building Leftist Heap

New leftist heap with one node

Moves in Building Leftist Heap

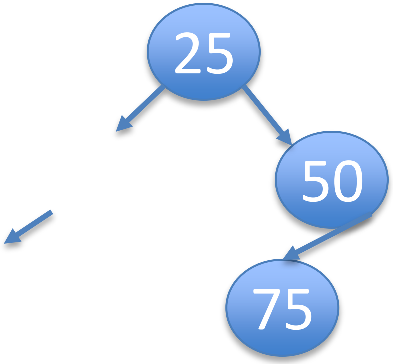


Normal insertion of a new node into the tree value 75.

First placed as far **right** as possible.

Then swung left to satisfy npls.

Moves in Building Leftist Heap

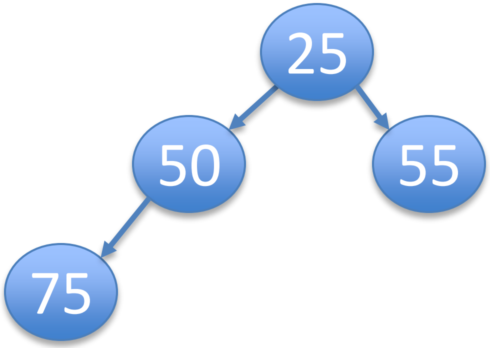


Normal insertion of a new node into the tree value 25.

As this is a min Tree, 25 is the new root.

Then swung left to satisfy npls.

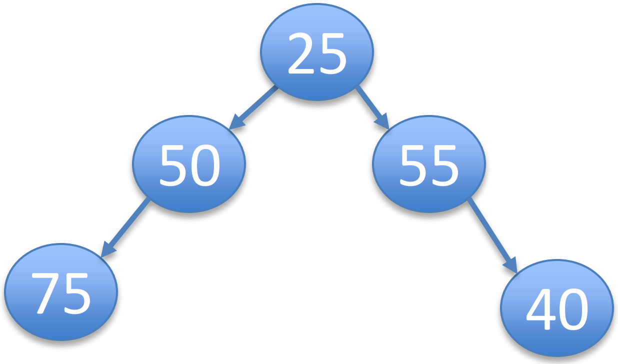
Moves in Building Leftist Heap



Normal insertion of a new node into the tree value 55.

No swing required.

Moves in Building Leftist Heap

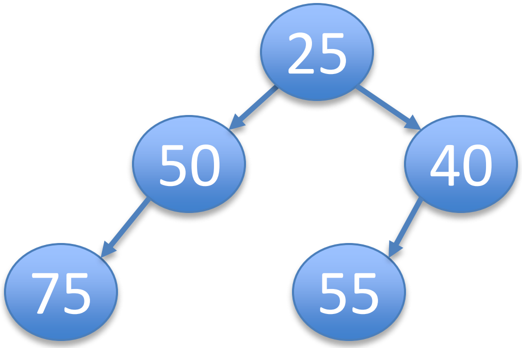


Normal insertion of a new node into the tree value 40.

Not a min heap at this point. Need to swap 40 and 55 and swing.

What is wrong with this?

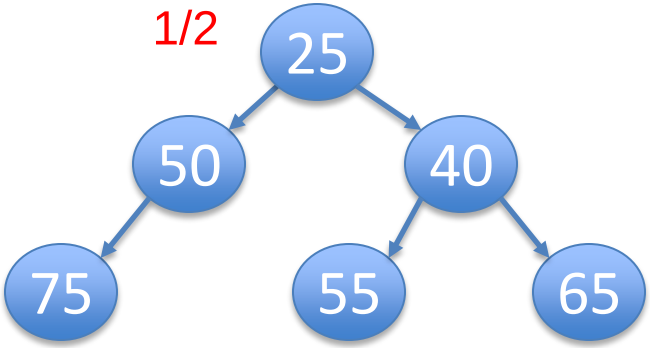
Moves in Building Leftist Heap



Normal insertion of a new node into the tree value 40.

Not a min heap at this point. Need to swap 40 and 55 and swing.

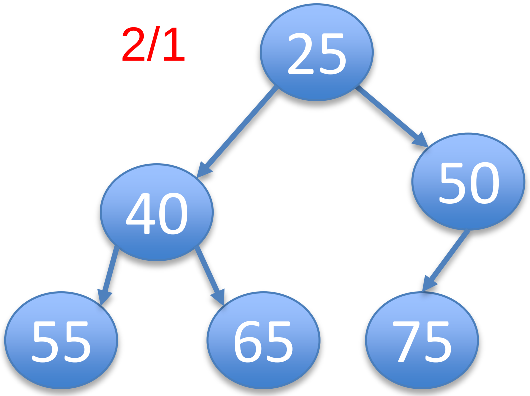
Moves in Building Leftist Heap



Normal insertion of a new node into the tree value 65.

While this is still a min heap, the npl at the root is not leftist

Moves in Building Leftist Heap



We need change this from 1/2 to 2/1 so that it remains leftist.

To do this, we switch the left and the right subtrees.

After we do the swap, the npl of the root is compliant.

Leftist Heap Algorithm

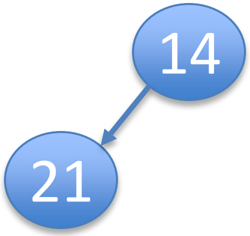
- Add new node to right-side of tree, in order
- If new node is to be inserted as a parent (parent < children)
  - make new node parent
  - link children to it
  - link grandparent down to new node **(now new parent)**
- If leaf, attach to right of parent
- If no left sibling, push to left (hence left-ist)
  - why?? (answer in a second)
- Else left node is present, leave at right child
- Update all ancestors' npls
- Check each time that all nodes left npl > = right npls
  - if not, swap children or node where this condition exists

Building a Leftist Heap Example

~~21~~, 14, 17, 10, 3, 23, 26, 8

Building a Leftist Heap Example

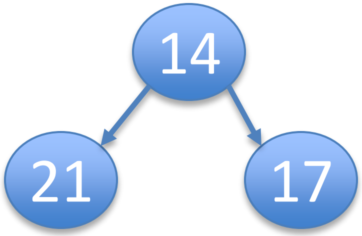
~~21~~, ~~14~~, 17, 10, 3, 23, 26, 8



Insert 14 as the new root

Building a Leftist Heap Example

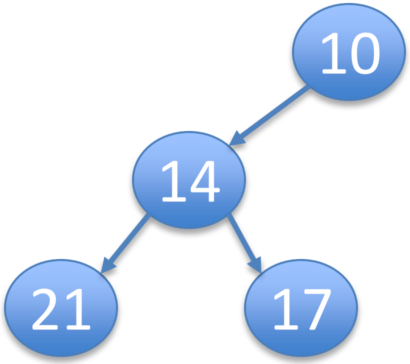
~~21~~, ~~14~~, ~~17~~, 10, 3, 23, 26, 8



Insert 17 as the right child of 14

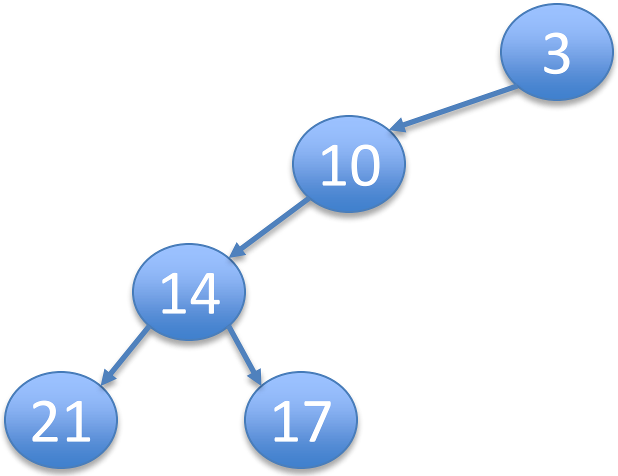
Building a Leftist Heap Example

~~21~~, ~~14~~, ~~17~~, ~~10~~, 3, 23, 26, 8



Insert 10 as the new root

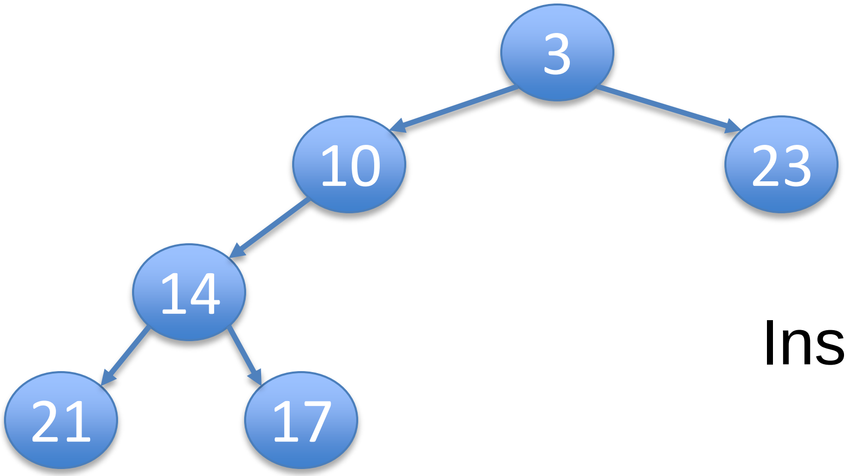
Building a Leftist Heap Example



~~21~~, ~~14~~, ~~17~~, ~~10~~, ~~3~~, 23, 26, 8

Insert 3 as the new root

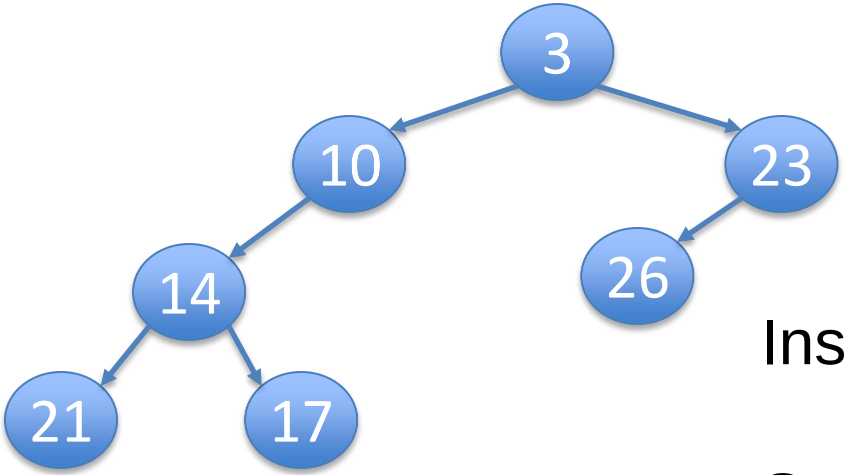
Building a Leftist Heap Example



~~21~~, ~~14~~, ~~17~~, ~~10~~, ~~3~~, ~~23~~, 26, 8

Insert 23 as the right child of 3

Building a Leftist Heap Example



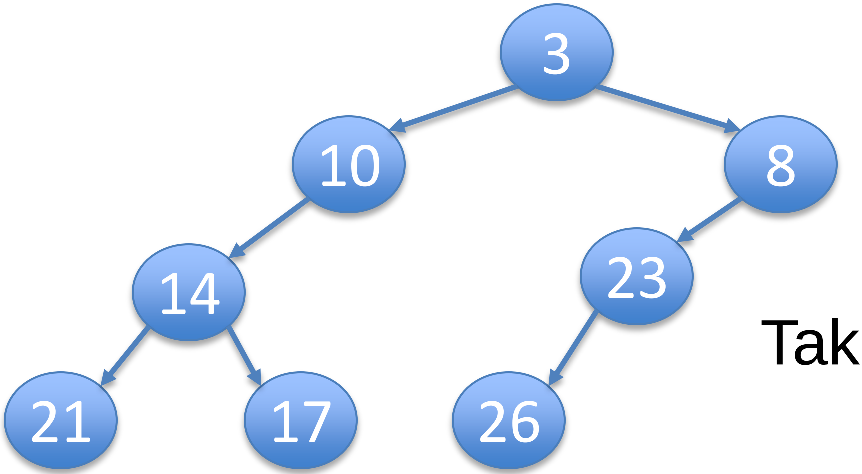
~~21~~, ~~14~~, ~~17~~, ~~10~~, ~~3~~, ~~23~~, ~~26~~, 8

Insert 26 as the right child of 23

Swing 26 to the left

Building a Leftist Heap Example

~~21~~, ~~14~~, ~~17~~, ~~10~~, ~~3~~, ~~23~~,
~~26~~, ~~8~~



Take the right subtree of root 3: nodes 23 & 26

Insert 8 as the new root (parent of 23)

Reattach to original root

Merging Leftist Heaps

Merging Leftist Heaps

- In the code, adding a single node is treated as merging a heap (just one node) with an established heap's root
  - And work from that root as we just went over
- We will go over merging whole heaps momentarily
- But in reality, isn't ONE node a heap??

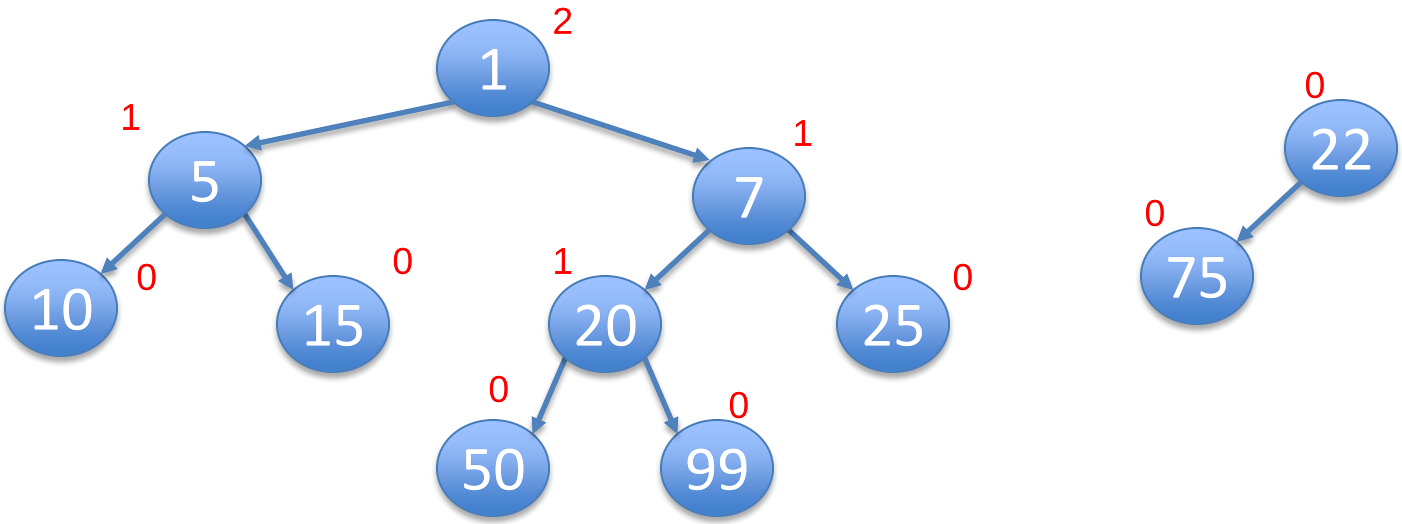
Merging Leftist Heaps

- The heaps we are about to merge must be LEFTIST heaps
- At the end we will get a heap that is both
  - a min-heap
  - leftist

Merging Leftist Heaps

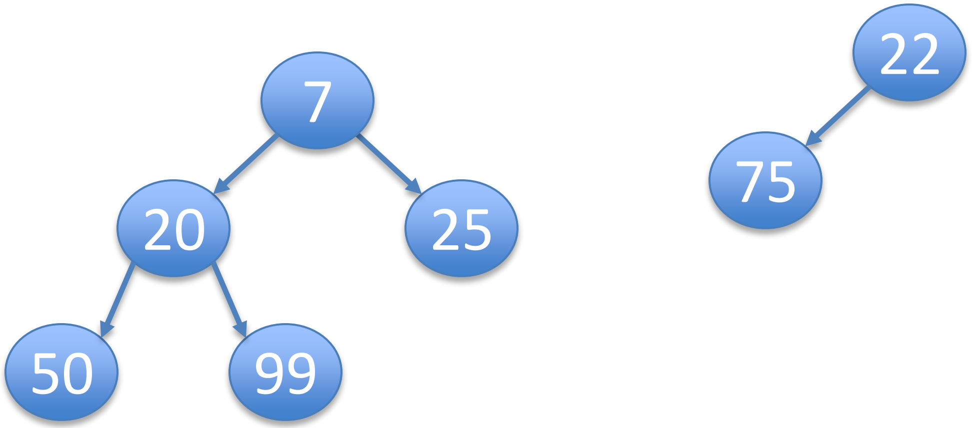
- The Merge procedure takes two leftist trees, A and B, and returns a leftist tree that contains the union of the elements of A and B. In a program, a leftist tree is represented by a pointer to its root.

Merging Leftist Heaps Example



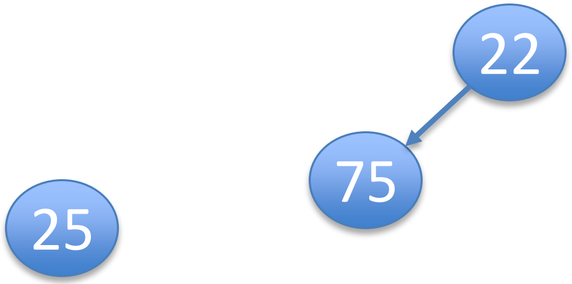
Where should we attempt to merge?

Merging Leftist Heaps Example



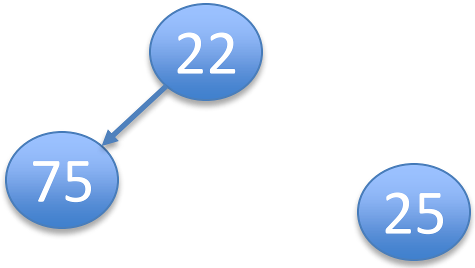
In the right sub-tree

Merging Leftist Heaps Example



All the way down to the right most node

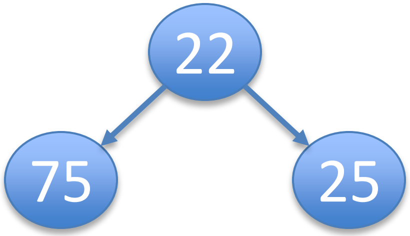
Merging Leftist Heaps Example



As there are two nodes in the right subtree, swap.

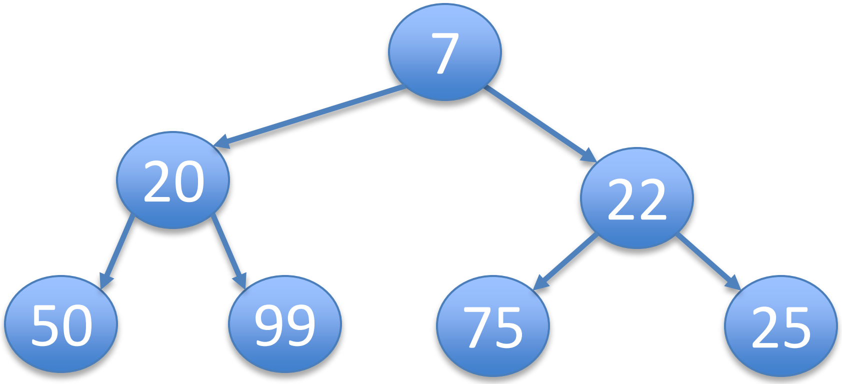
Important: We don't "split" a heap, so 22 must be the parent in this merge

Merging Leftist Heaps Example



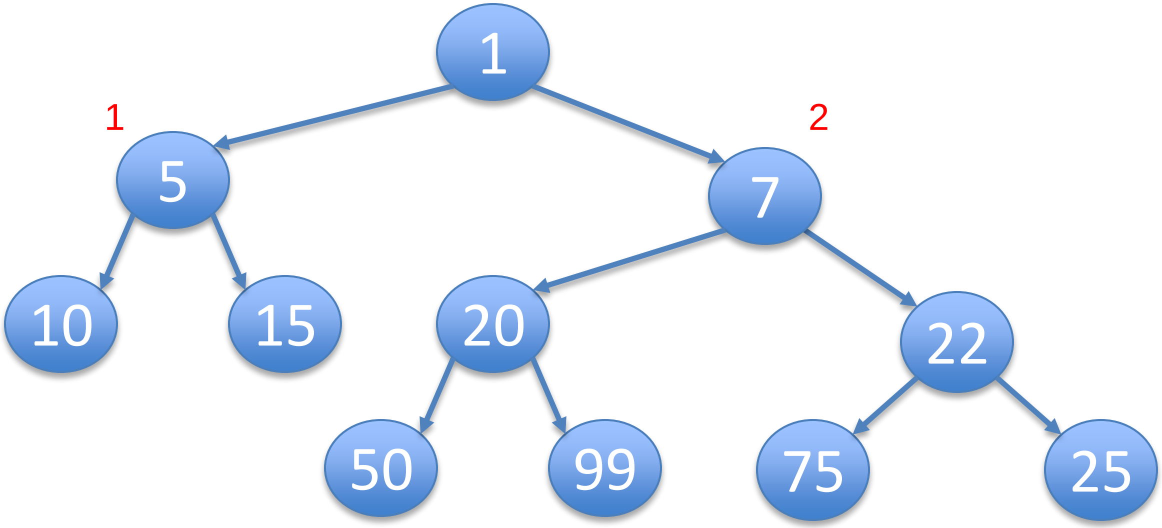
Merge two subtrees

Merging Leftist Heaps Example



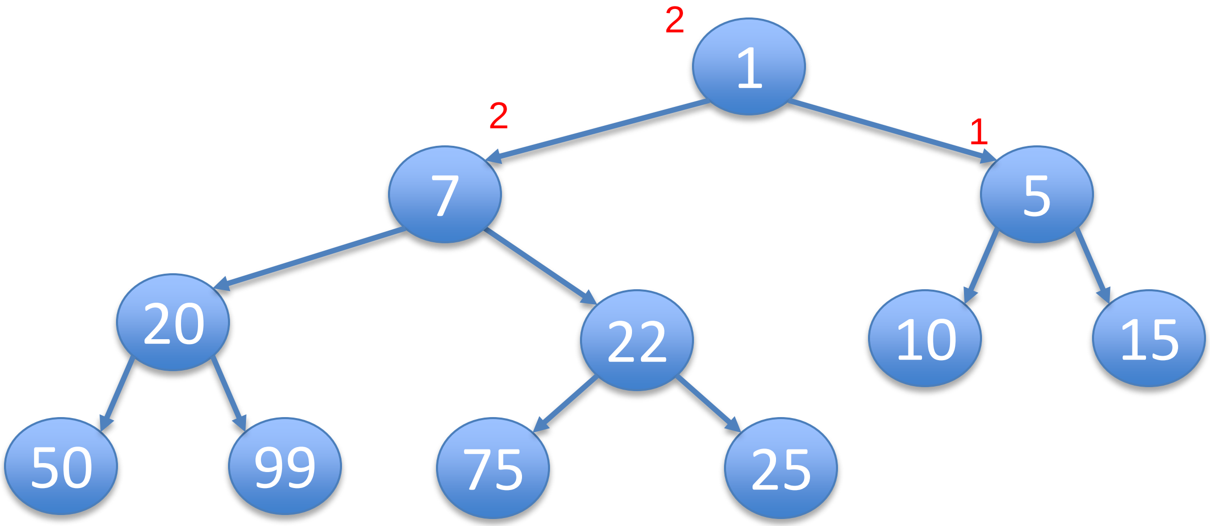
Next level of the tree

Merging Leftist Heaps Example



Right side of the tree has a npl of 2 so we need to swap

Merging Leftist Heaps Example



Now the highest npl is on the left.

Merging Leftist Heaps

- – Start at the (sub) root, and finalize the node AND LEFT with the smallest value
- – REPEADLY, until no lists left unmerged.
  - Start at the **rightmost** root of the sub-tree, and finalize the node AND LEFT with the **next** smallest value in leftist lists.
  - Add to RIGHT of finalized tree.
- – Verify that it is a Min Heap!! (Parent < Children)
- – Verify a leftist heap! (left npl >= right npl)
  - if not, swap troubled node with sibling

Merging Leftist Heaps Code

```
/**
    * Merge rhs into the priority queue.
    * rhs becomes empty. rhs must be different from this.
    */
   void merge( LeftistHeap & rhs )
   {
       if( this == &rhs )     // Avoid aliasing problems
           return;

       root = merge( root, rhs.root );
       rhs.root = NULL;
   }
```

Merging Leftist Heaps Code

```
/**
 * Internal method to merge two roots.
 * Deals with deviant cases and calls recursive merge1.
 */
LeftistNode * merge( LeftistNode *h1, LeftistNode *h2 )
{
    if( h1 == NULL )
        return h2;
    if( h2 == NULL )
        return h1;
    if( h1->element < h2->element )
        return merge1( h1, h2 );
    else
        return merge1( h2, h1 );
}
```

Merging Leftist Heaps Code

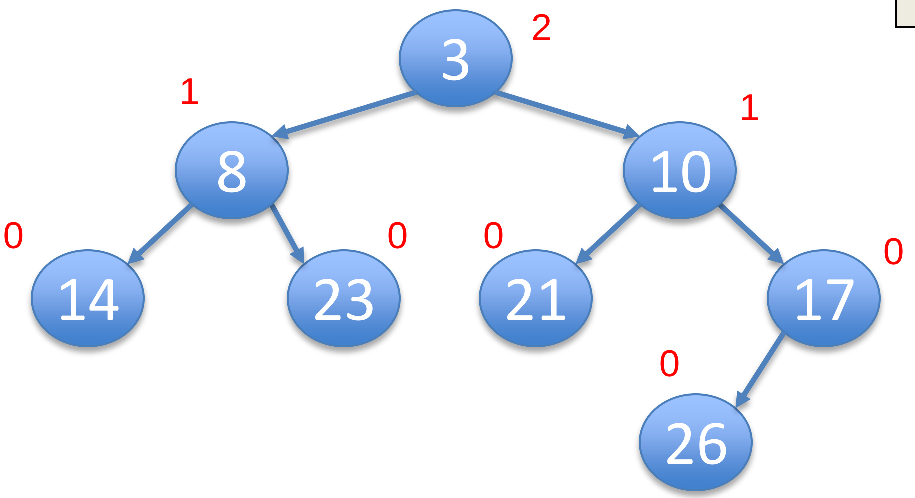
```
/**
 * Internal method to merge two roots.
 * Assumes trees are not empty, & h1's root contains smallest item.
 */
LeftistNode * merge1( LeftistNode *h1, LeftistNode *h2 )
{
    if( h1->left == NULL ) // Single node
        h1->left = h2;     // Other fields in h1 already accurate
    else
    {
        h1->right = merge( h1->right, h2 );
        if( h1->left->npl < h1->right->npl )
            swapChildren( h1 );
        h1->npl = h1->right->npl + 1;
    }
    return h1;
}
```

Deleting from Leftist Heap

Deleting from Leftist Heap

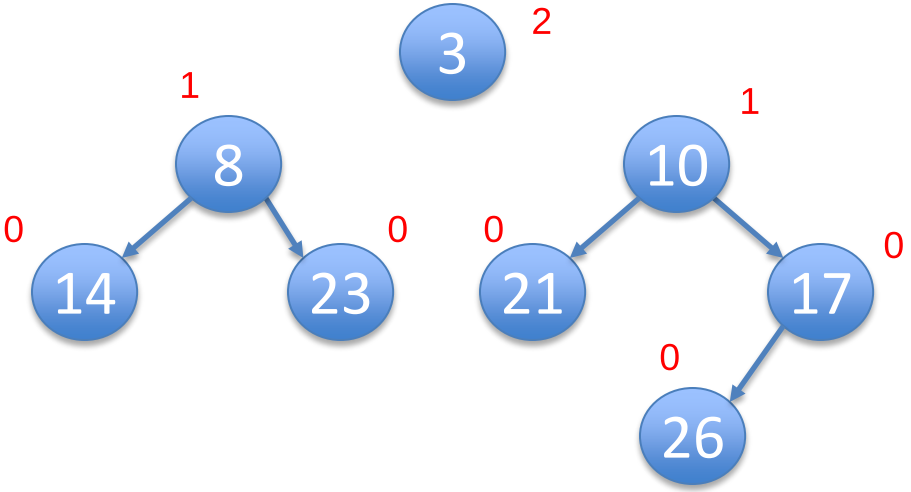
- Simple to just remove a node (since at top)
  - this will make two trees
- Merge the two trees like we just did

Deleting from Leftist Heap



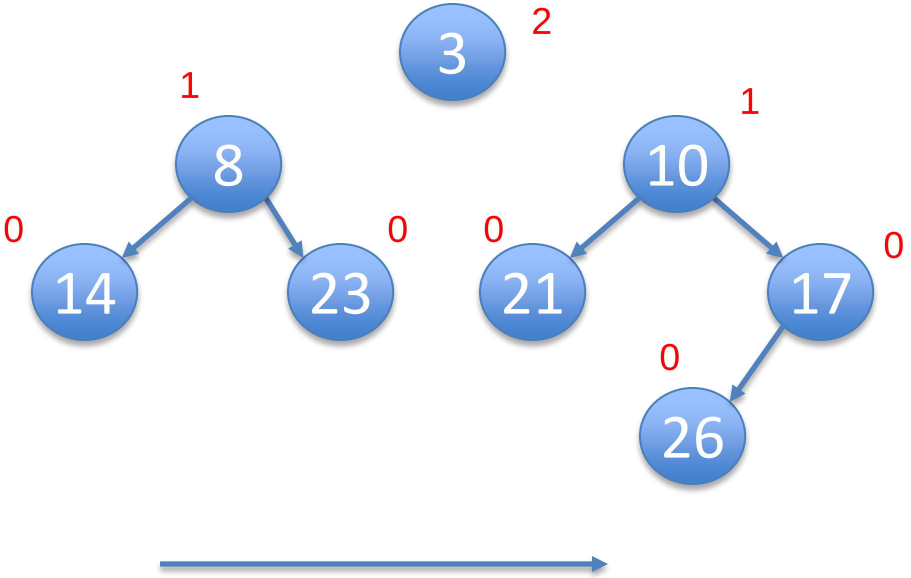
We remove the root.

Deleting from Leftist Heap



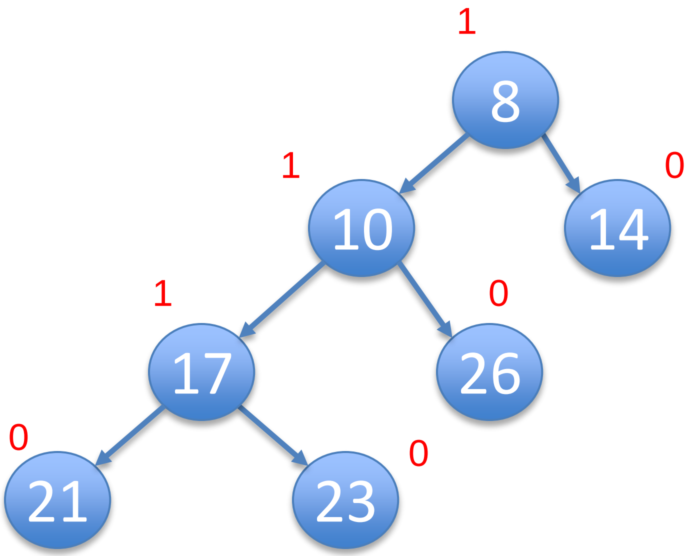
Then we do a merge and because min is in left subtree, we recursively merge right into left

Deleting from Leftist Heap



Then we do a merge and because min is in left subtree, we recursively merge right into left

Deleting from Leftist Heap



After Merge

Leftist Heaps

- Merge with two trees of size n
  - O(log n), we are not creating a totally new tree!!
  - some was used as the LEFT side!
- Inserting into a left-ist heap
  - O(log n)
  - same as before with a regular heap
- `deleteMin` with heap size n
  - O(log n)
  - remove and return root (minimum value)
  - merge left and right subtrees

Announcements

- Project 3
  - – Due Tuesday, November 7th at 8:59:59 PM