

# CSE 326 Data Structures

CSE 326 : Dave Bacon

Priority Queues : Leftist heaps,  
Skew heaps

# Logistics

- **Homework 2 due in folder at front of class!**  
**(I'll pick up ~10min into class)**
- **Final exam, Thursday March 15**  
**Section A: (Bacon) 8:30 - 10:20 MGH 231**  
Problem? Email me!
- Reading 6.8, Start Chapter 4!

Can do in  $\Theta(\log n)$  worst case time.

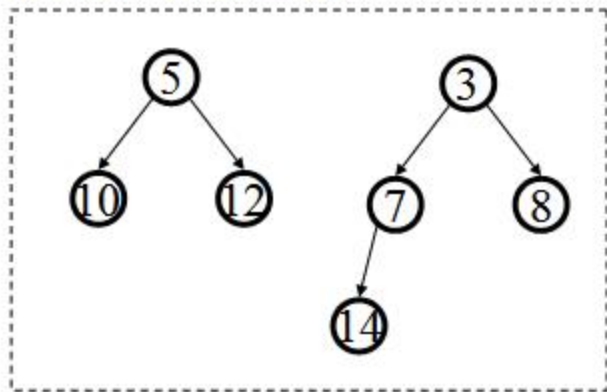
# Tablets!

- The future is now! Draw a picture of something from Seattle (both use pen):

Can do in  $\Theta(\log n)$  worst case time.

# A Finicky Operation

- Merge two heaps.
- Your Idea:



Can do in  $\Theta(\log n)$  worst case time.

Your Running Time.

# Leftist Heaps

Idea:

Focus all heap maintenance work in one small part of the heap

Leftist heaps: Actually TRIES to be unbalanced.

1. Most nodes are on the left
2. All the merging work is done on the right

# Definition: Null Path Length

*null path length (npl)* of a node  $x$  = the number of nodes between  $x$  and a null in its subtree

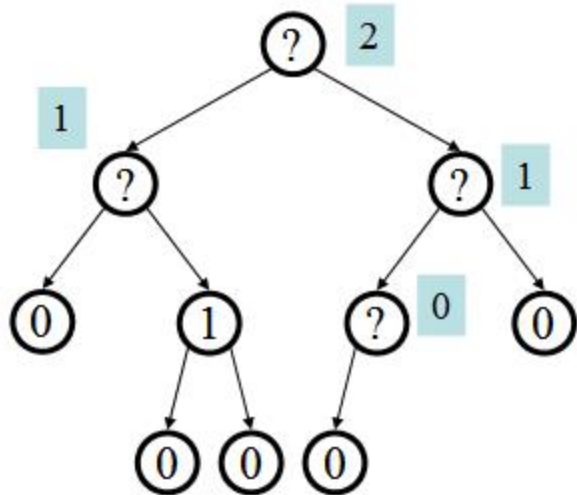
OR

$npl(x)$  = min distance to a descendant with 0 or 1 children

- $npl(\text{null}) = -1$
- $npl(\text{leaf}) = 0$
- $npl(\text{single-child node}) = 0$

Equivalent definitions:

1.  $npl(x)$  is the height of largest complete subtree rooted at  $x$
2.  $npl(x) = 1 + \min\{npl(\text{left}(x)), npl(\text{right}(x))\}$



# Leftist Heap Properties

- Heap-order property
  - parent's priority value is  $\leq$  to children's priority values
  - result: minimum element is at the root
- Leftist property
  - For every node  $x$ ,  $npl(\text{left}(x)) \geq npl(\text{right}(x))$
  - result: tree is at least as "heavy" on the left as the right

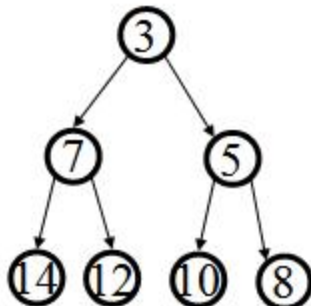
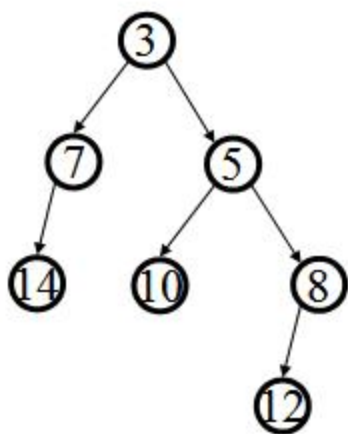
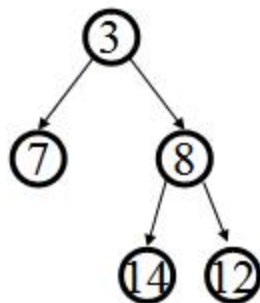
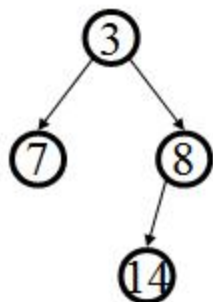
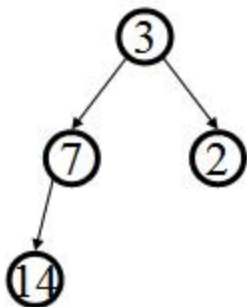
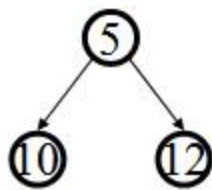
Are leftist trees...

complete?

balanced?

# Leftist Heaps?

Circle the Leftist Heaps



Rafael Correa  
Ecuador

## Leftist Heap Size

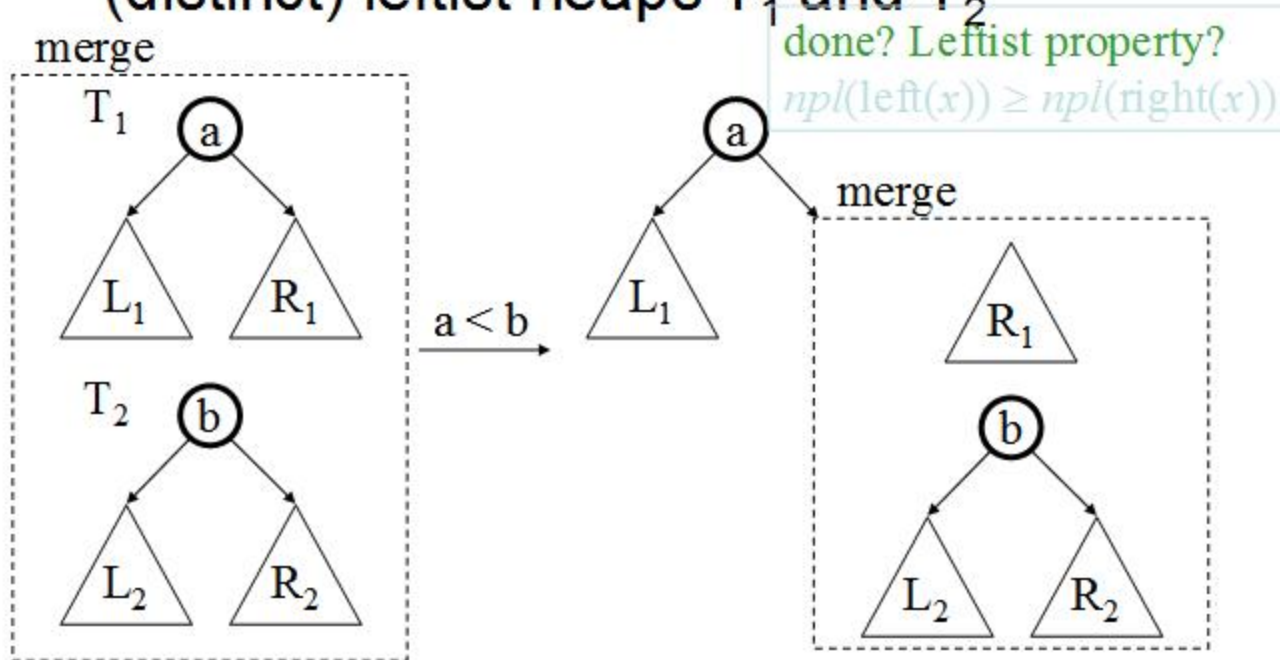
- A leftist tree with  $r$  nodes on the right path must have at least  $2^r - 1$  nodes
- $r=1$
- Assume true for  $1, \dots, r-1$ . Then leftist heap size  $r$ :

## Merge two leftist heaps (basic idea)

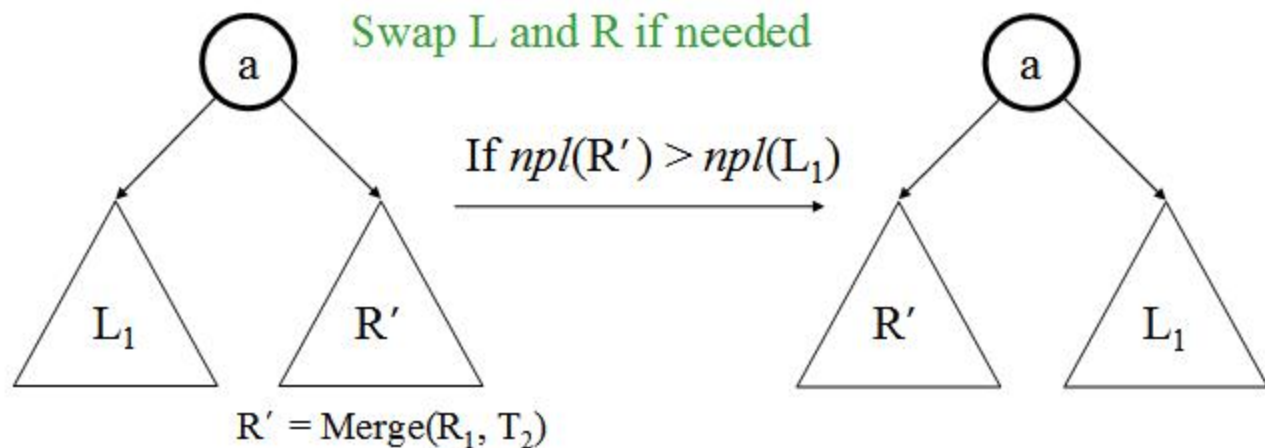
- Put the smaller root as the new root,
- Hang its left subtree on the left.
- Recursively merge its right subtree and the other tree.

# Merging Two Leftist Heaps

- $\text{merge}(T_1, T_2)$  returns one leftist heap containing all elements of the two (distinct) leftist heaps  $T_1$  and  $T_2$



# Leftist Merge Continued



Work at each step = call to merge, swap (constant)

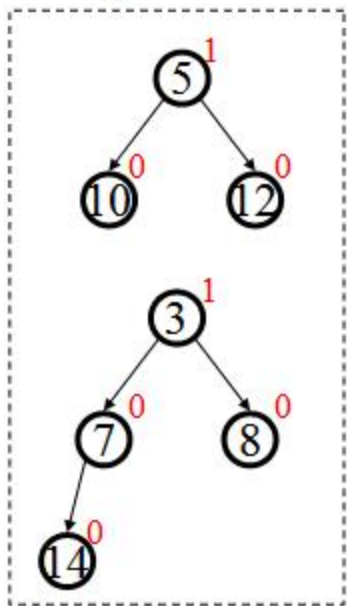
traverse the right path of both trees = length is at most  $\log N$

runtime:

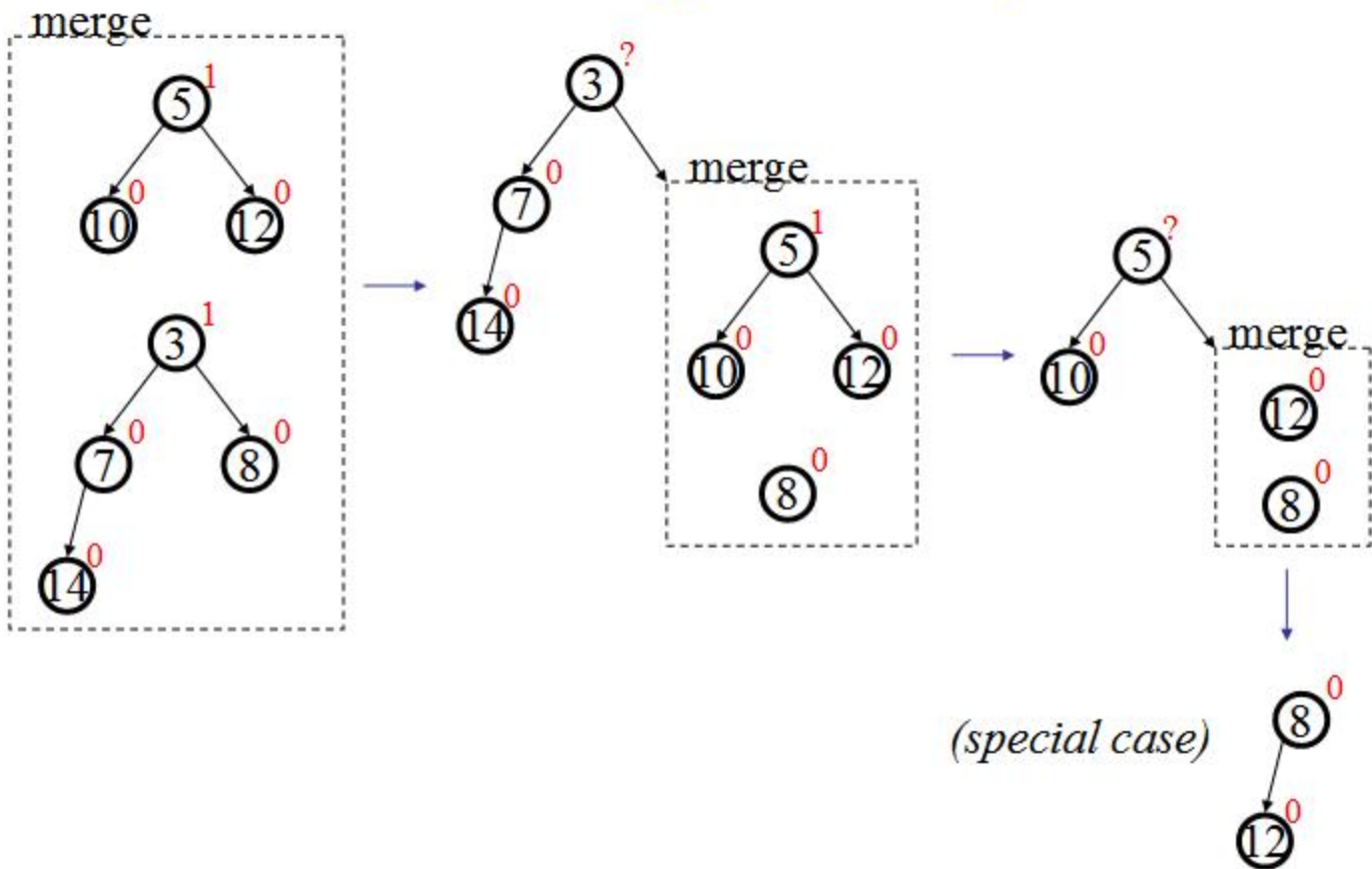
$O(\log n)$

# Leftist Merge: Do It!

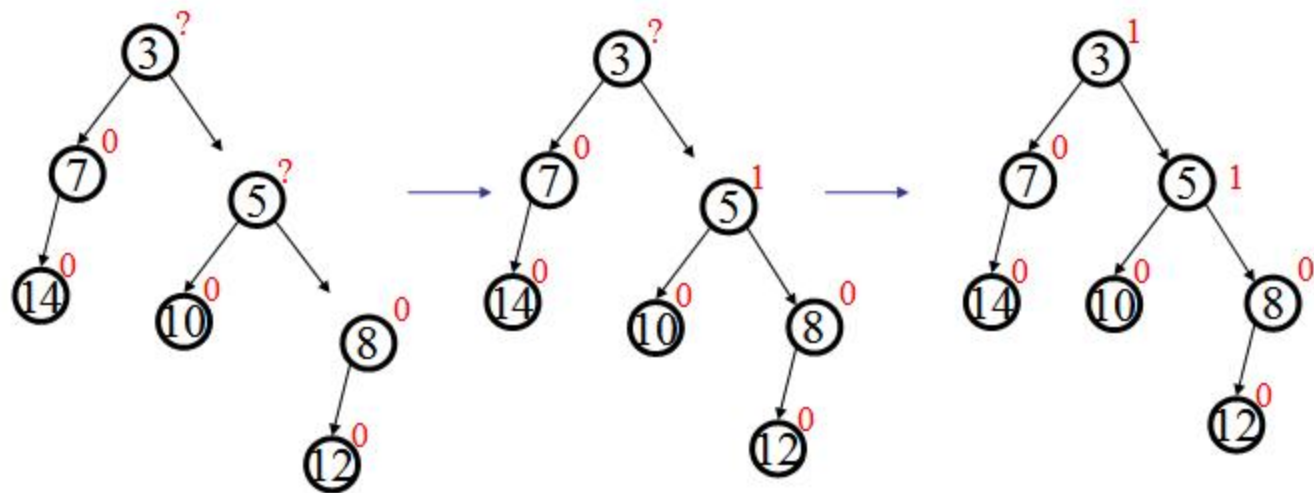
merge



# Leftist Merge Example



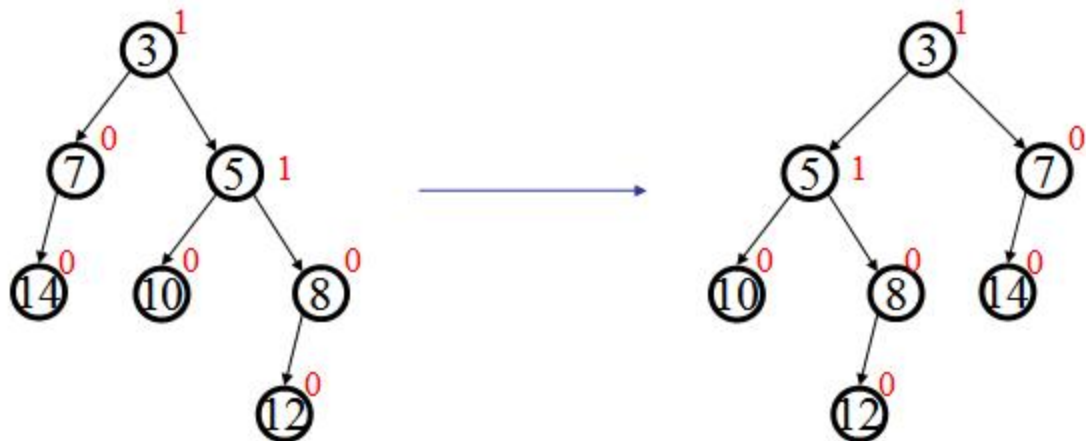
# Sewing Up the Leftist Example



Done?

We forgot to swap L-R at places!

## Finally...(Leftist)



# Operations on Leftist Heaps

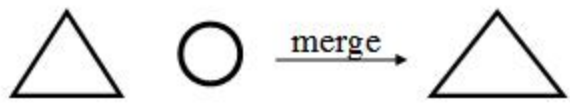
Use merge to do everything

merge with two trees of total size  $n$ :  $O(\log n)$

insert with heap size  $n$ :  $O(\log n)$

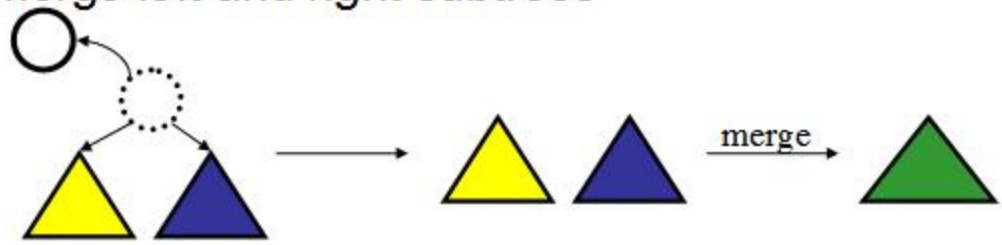
- pretend node is a size 1 leftist heap
- insert by merging original heap with one node heap

Is this true? yes



deleteMin with heap size  $n$ :  $O(\log n)$

- remove and return root
- merge left and right subtrees



# Random Definition: Amortized Time

am·or·tized time:

**Running time limit resulting from “writing off” expensive runs of an algorithm over multiple cheap runs of the algorithm, usually resulting in a lower overall running time than indicated by the worst possible case.**

If  $M$  operations take total  $O(M \log N)$  time,  
*amortized* time per operation is  $O(\log N)$

Difference from **average time**:

Average - given average input, here is the result. Still exist bad sequences.

Amortized - given *any* input, we have this guarantee.

# Skew Heaps

## Problems with leftist heaps

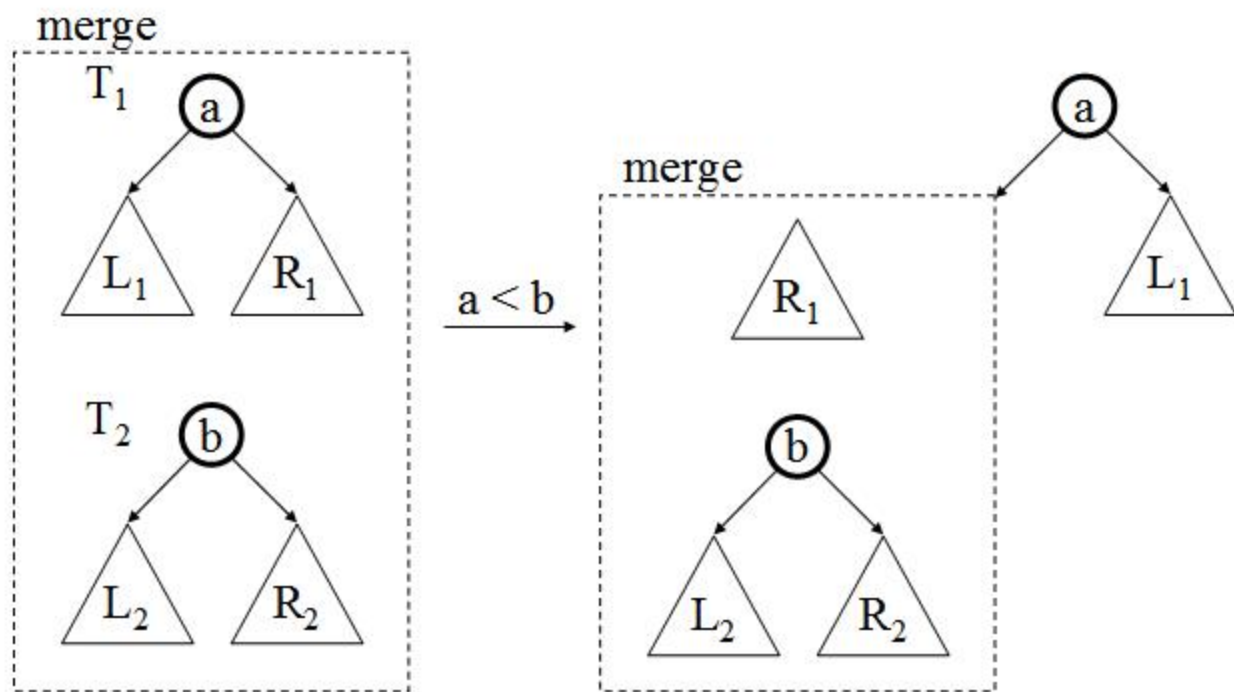
- Simple to implement,
- no npl stuff

- extra storage for npl
- extra complexity/logic to maintain and check npl
- right side is “often” heavy and requires a switch

## Solution: skew heaps

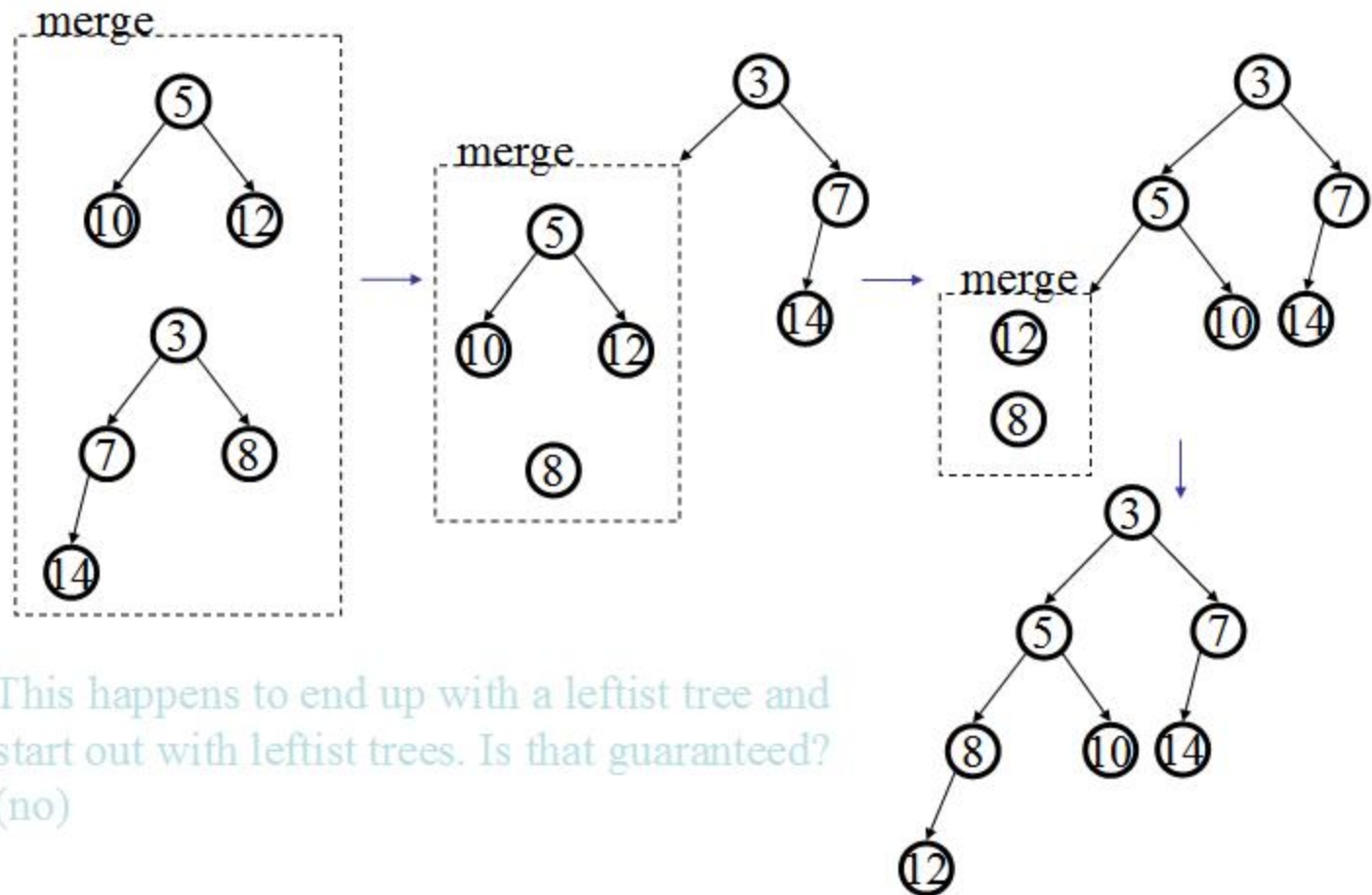
- “blindly” adjusting version of leftist heaps
- merge *always* switches children when fixing right path
- amortized time for: merge, insert, deleteMin =  $O(\log n)$
- however, worst case time for all three =  $O(n)$

# Merging Two Skew Heaps



**Only one step per iteration, with children *always* switched**

# Example



# Skew Heap Code

```
void merge(heap1, heap2) {
    case {
        heap1 == NULL: return heap2;
        heap2 == NULL: return heap1;
        heap1.findMin() < heap2.findMin():
            temp = heap1.right;
            heap1.right = heap1.left;
            heap1.left = merge(heap2, temp);
            return heap1;
        otherwise:
            return merge(heap2, heap1);
    }
}
```

# Runtime Analysis: Worst-case and Amortized

- No worst case guarantee on right path length!
- All operations rely on merge

⇒ worst case complexity of all ops =  $O(n)$

- Will do amortized analysis later in the course  
(see chapter 11 if curious)
- Result:  $M$  merges take time  $M \log n$

⇒ amortized complexity of all ops =  $O(\log n)$

# Comparing Heaps

- simple,
- memory efficient,
- $O(\log N)$  ins/delete
- cannot merge quickly.

- supports merge quickly.
- $O(\log n)$  for others but... Practically, extra storage, link traversals and recursion make it bigger and slower

- Binary Heaps

- Leftist Heaps

- d-Heaps

- Insert is faster than binary heaps.
- delete in slightly slower
- Similar to binheaps

- Skew Heaps

- slightly less storage,
- somewhat faster and simpler than leftist.
- Better amortized perf.

## Still scope for improvement!

Leftist and skew: all ops in  $O(\log N)$ , BQs: worst =  $O(\log N)$ , but insert in  $O(1)$  on average

# Assessment

- The pace of the course material is  
(Way too fast, Too fast, Just right, Too slow, Way too slow)  
(Way too fast, Too fast, Just right, Too slow, Way too slow)
- The difficulty of the homeworks have been  
(Way too hard, Too hard, Just right, Too easy, Way too easy)  
(Way too hard, Too hard, Just right, Too easy, Way too easy)
- The lectures have been  
(Useful, Somewhat Useful, Somewhat Useless, Useless)  
(Useful, Somewhat Useful, Somewhat Useless, Useless)