CSE 326: Data Structures
Topic #4
Putting Our Heaps Together

Luke McDowell
Summer Quarter 2003

Outline

- Finish Binary Heaps
- D-heaps
- Leftist Heaps
- Skew Heaps
- Comparing Heaps

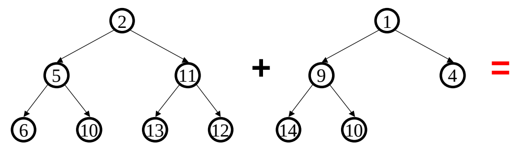
New Operation: Merge

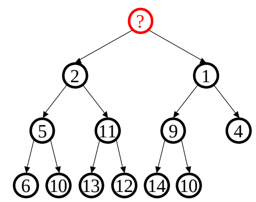
Given two heaps, merge them into one heap

- first attempt: insert each element of the smaller heap into the larger.
  runtime:
- second attempt: concatenate heaps' arrays and run buildHeap.
  runtime:

How about O(log n) time?

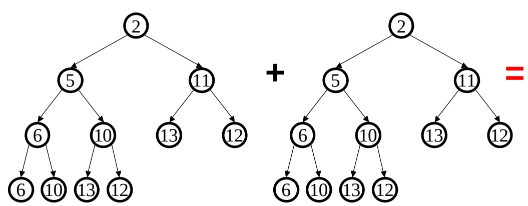
Idea: Hang a New Tree





Now, just percolate down!

Idea: Hang a New Tree



Problem?

Leftist Heaps

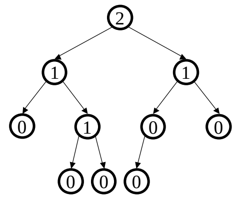
- Idea:

  make it so that all the work you have to do in maintaining a heap is in one small part
- Leftist heap:
  - almost all nodes are on the left
  - all the merging work is on the right

Random Definition: Null Path Length

the *null path length (npl)* of a node is the number of nodes between it and a null in the tree

- npl(null) = -1
- npl(leaf) = 0
- npl(single-child node) = 0



another way of looking at it: npl is the height of complete subtree rooted at this node

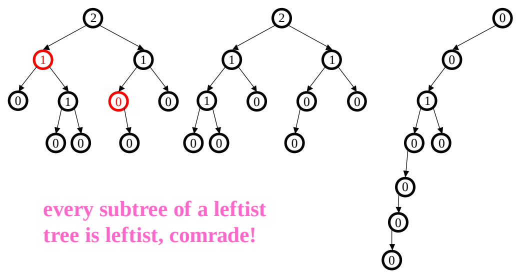
Leftist Heap Properties

- Heap-order property
  - parent's priority value is ? to childrens' priority values
  - result: minimum element is at the root
- Leftist property
  - null path length of left subtree is ? npl of right subtree
  - result: tree is at least as "heavy" on the left as the right

Are leftist trees…
complete?
balanced?

Leftist tree examples

NOT leftist — leftist — leftist



every subtree of a leftist tree is leftist, comrade!

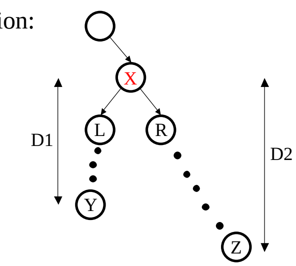
Right Path in a Leftist Tree is Short (#1)

- Claim: The right path is as short as *any* in the tree
- Proof by contradiction:

Shorter path: D1 < D2

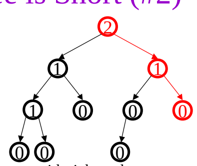
Npl(left):

Npl(right):



Right Path in a Leftist Tree is Short (#2)

- Claim: If the right path has length at least **r**, the tree has at least $2^r - 1$ nodes
- Proof by induction



Basis: **r = 1**. Tree has at least one node: $2^1 - 1 = 1$

Inductive step: assume true for **r' < r**. Prove for tree with right path >= r.

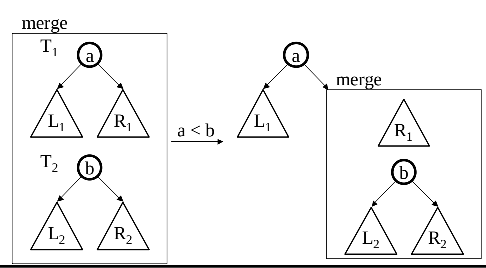
1. Right subtree: right path of at least r - 1 nodes ✍ $2^{r-1} - 1$ subtree nodes (induction)
2. Left subtree: also right path of at least **r - 1** ✍ $2^{r-1} - 1$ subtree nodes (induction + from the preceding theorem)
3. Root: ✍ 1 node

Total: $2^{r-1} - 1 + 2^{r-1} - 1 + 1 = 2^r - 1$

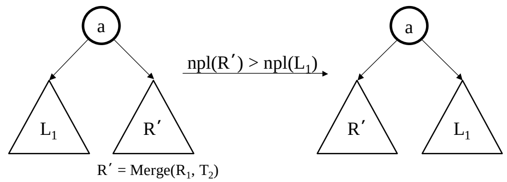
- So, a leftist tree with at least **n** nodes has a right path of at most **log n** nodes

Merging Two Leftist Heaps

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



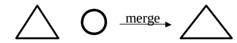
Merge Continued



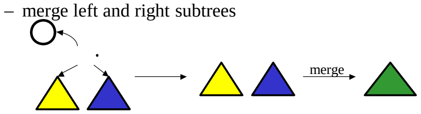
runtime:

Operations on Leftist Heaps

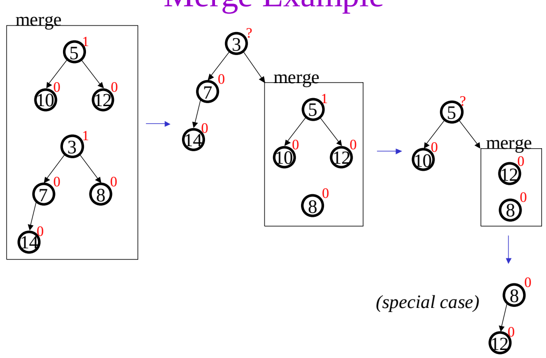
- merge with two trees of total size n: O(log n)
- insert with heap size n: O(log n)
  - pretend node is a size 1 leftist heap
  - insert by merging original heap with one node heap



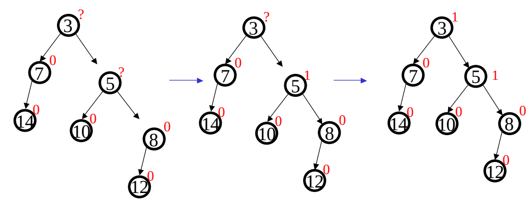
- deleteMin with heap size n: O(log n)
  - remove and return root
  - merge left and right subtrees



Merge Example

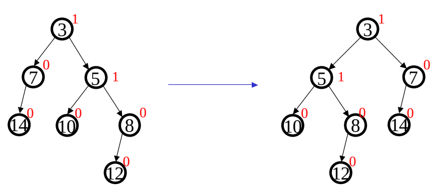


Sewing Up the Example



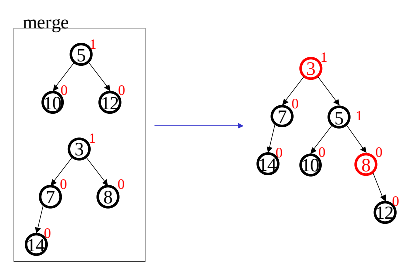
Done?

Finally…



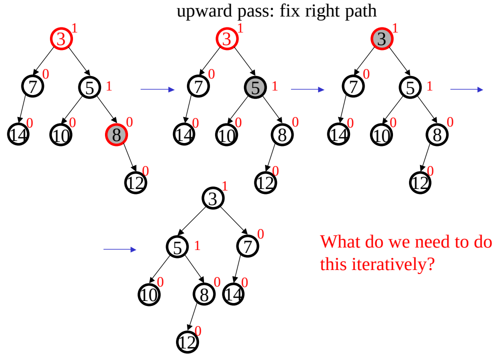
Iterative Leftist Merging

downward pass: merge right paths



Iterative Leftist Merging

upward pass: fix right path



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.

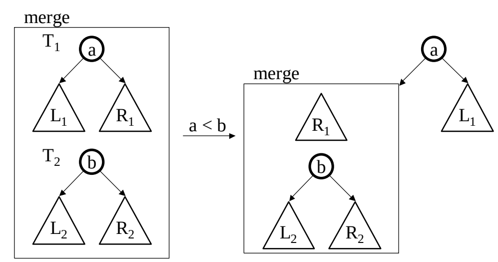
If M operations take total O(M log N) time, amortized time per operation is O(log N)

Difference from **average time:**

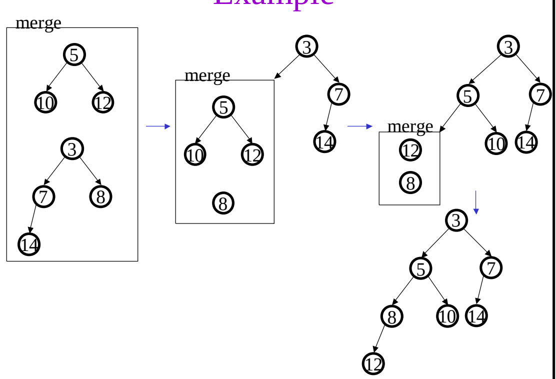
Skew Heaps

- Problems with leftist heaps
  - extra storage for npl
  - two pass merge (with stack!)
  - extra complexity/logic to maintain and check npl
- Solution: skew heaps
  - blind adjusting version of leftist heaps
  - amortized time for merge, insert, and deleteMin is O(log n)
  - worst case time for all three is O(n)
  - merge *always* switches children when fixing right path
  - iterative method has only one pass

Merging Two Skew Heaps



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);
  }
}
```

Comparing Heaps

- Binary Heaps
- Leftist Heaps
- d-Heaps
- Skew Heaps
- Binomial Queues

To Do

- Continue homework #2
  - Start early!
- Start chapter 4 in the book

Coming Up

- Dictionary ADT
- Self-Balancing Trees