CSE 326: Data Structures
Lecture #5
Political Heaps

Bart Niswonger

Summer Quarter 2001

Today's Outline

- Project comments & questions
- Things Bart Didn't Finish on Monday (Leftist Heaps)
- Skew Heaps
- Comparing Heaps

Merging Heaps

How can we make it fast?

- Array-based implementation:
- Pointer-based implementation:

Leftist Heaps

- Idea:

  make it so that all the work you have to do in maintaining a heap is in one small part

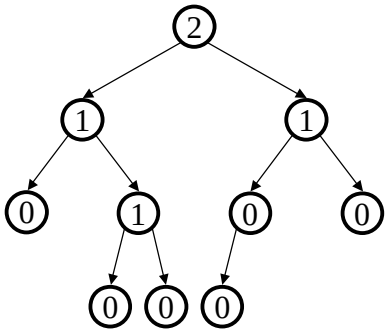
- Leftist heap:
  - almost all nodes are on the left
  - all the merging work is on the right

Not-so 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

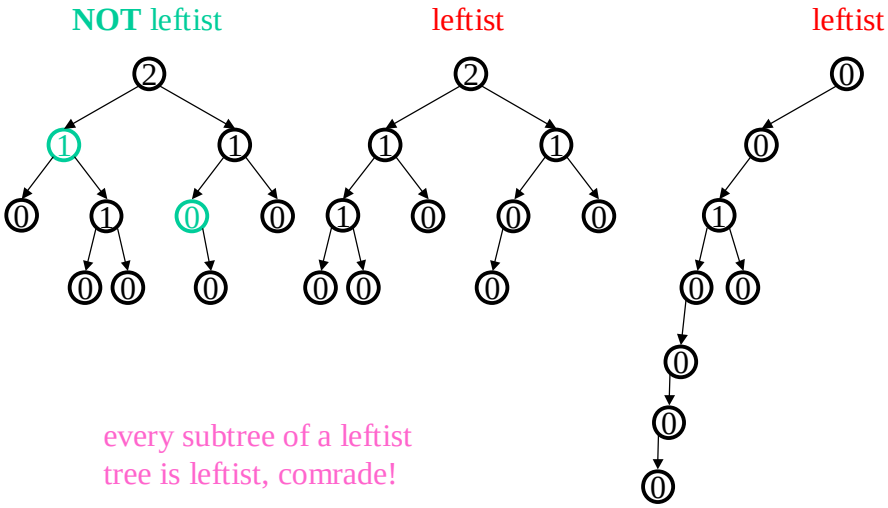


Leftist Heap Properties

- 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



Right Path in a Leftist Tree is Short

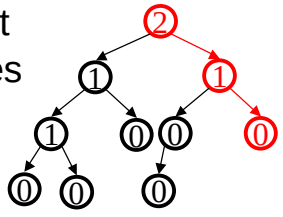
- If the right path has length at least **r**, the tree has at least $2^r - 1$ nodes
- Proof by induction

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

  Inductive step: assume true for $r' < r$. The right subtree has a right path of at least r - 1 nodes, so it has at least $2^{r-1} - 1$ nodes. The left subtree must also have a right path of at least $r - 1$ (otherwise, there is a null path of $r - 3$, less than the right subtree). Again, the left has $2^{r-1} - 1$ nodes. All told then, there are at least:

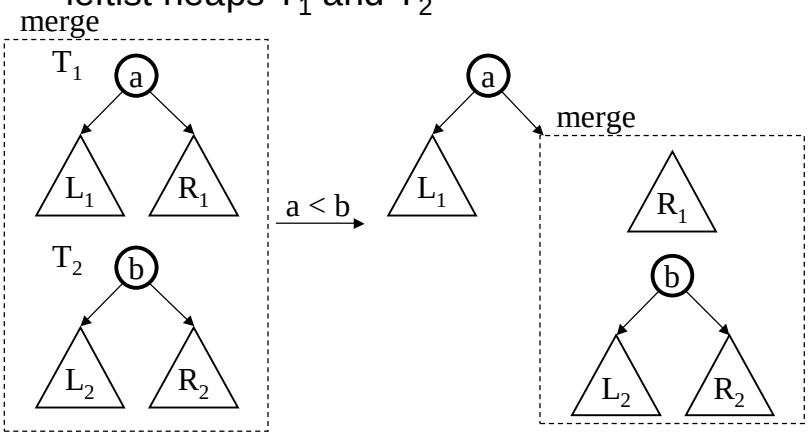
  $$2^{r-1} - 1 + 2^{r-1} - 1 + 1 = 2^r - 1$$

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

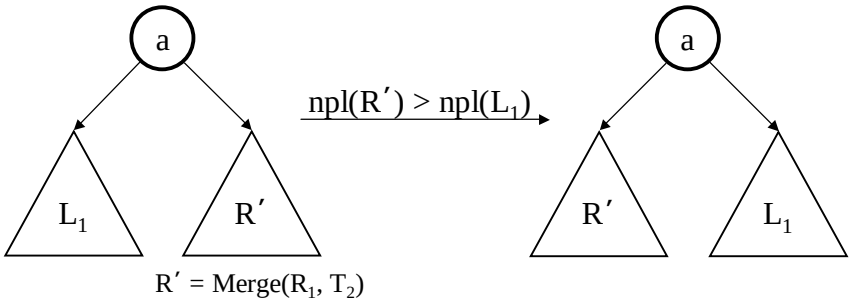


Merging Two Leftist Heaps

- 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



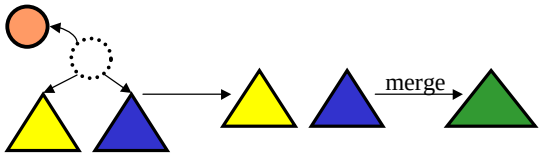
R′ = Merge(R_1, T_2)

runtime:

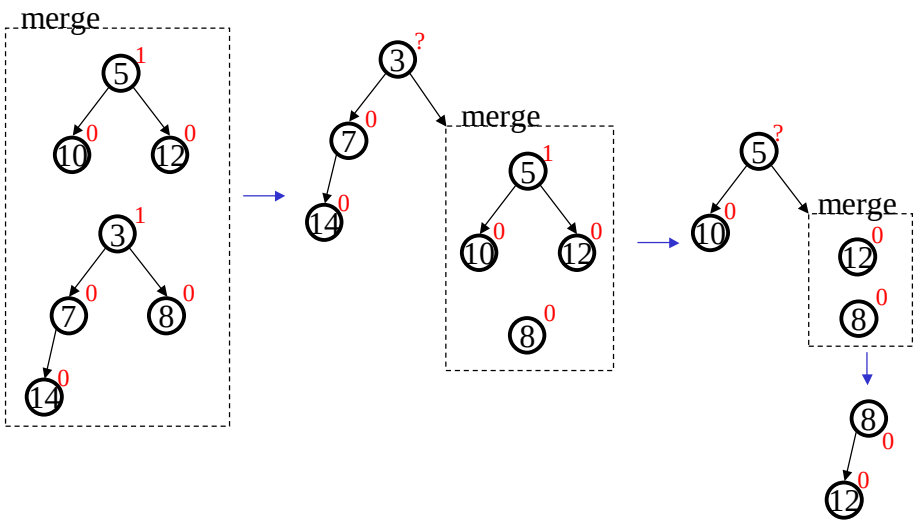
Operations on Leftist Heaps

- 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

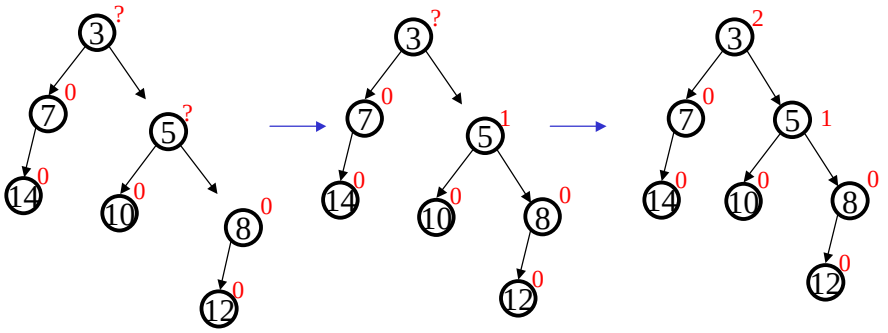
- deleteMin with heap size n: O(log n)
  - remove and return root
  - merge left and right subtrees



Example

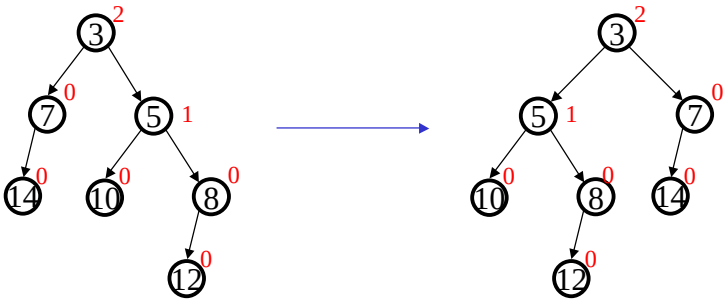


Sewing Up the Example



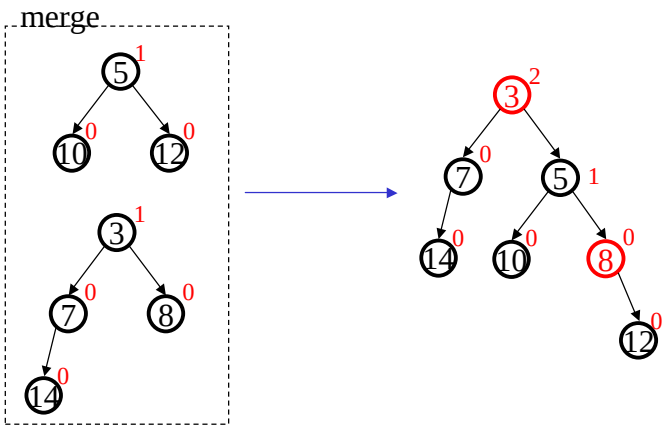
Done?

Finally…



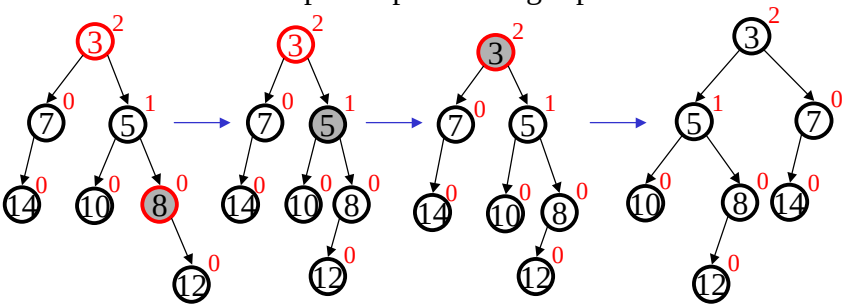
Iterative Leftist Merging

downward pass: merge right paths



Iterative Leftist Merging

upward pass: fix right path



What do we need to do this iteratively?

Random Definition: Amortized Time

am·or·tize
To write off an expenditure for (office equipment, for example) by prorating over a certain period.

time
A nonspatial continuum in which events occur in apparently irreversible succession from the past through the present to the future.

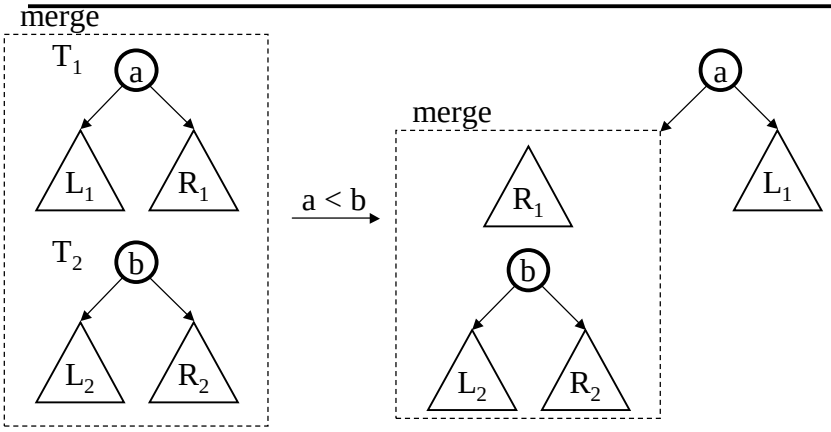
am·or·tized time
Running time limit resulting from writing off expensive runs of an algorithm over multiple cheap runs of the algorithm, usually resulting in a lower overall running time than indicated by the worst possible case.

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

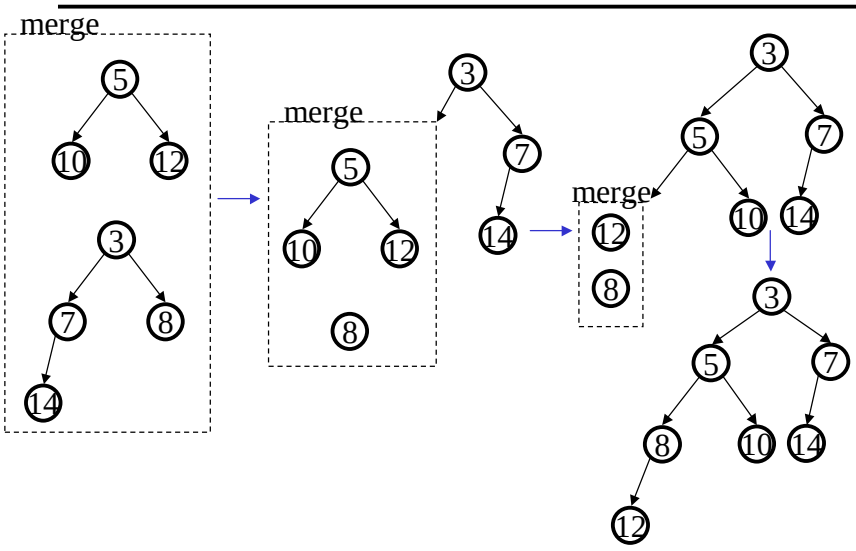
Skew Heaps

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

To Do

- Project I due tonight @ 10pm
  - Please email me if you are going to be late
  - Groups were due Monday
- Homework due tomorrow @ START of section

Coming Up

- Section
  - tomorrow: The Ashish Experience
- Project II
- Asymptotic Complexity!