Priority Queues

CSE 373
Data Structures & Algorithms
Ruth Anderson

Today's Outline

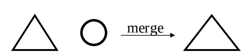
- **Announcements**
  - Assignment #3 due Thurs, Feb 5th.
- **Today's Topics:**
  - **Priority Queues**
    - **Leftist Heaps**
    - **Skew Heaps**

Other Heap Operations

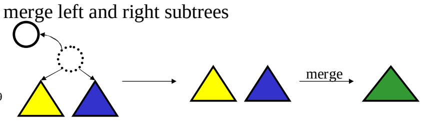
- insert ?
- deleteMin ?

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



Leftist Heaps: Summary

Good
-
-

Bad
-
-

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

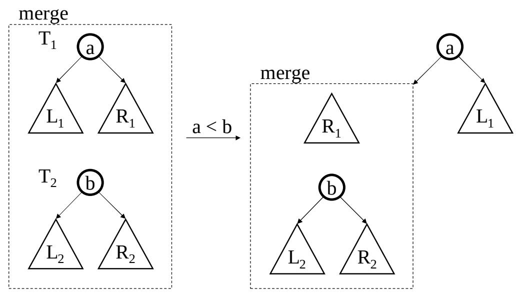
Problems with leftist heaps

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

Solution: skew heaps

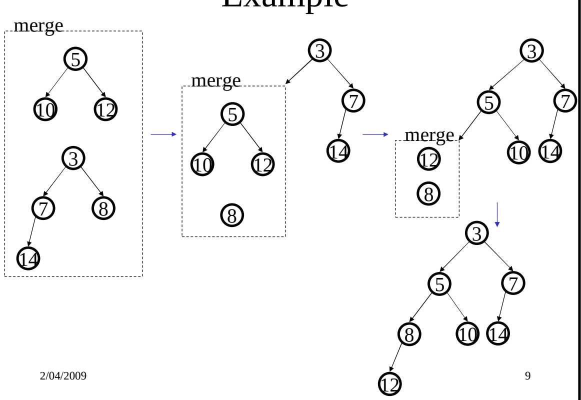
- "blindly" adjusting version of leftist heaps
- merge *always* switches children when fixing right path
- amortized time for: merge, insert, deleteMin = O(log *n*)
- however, worst case time for all three = O(*n*)

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 =

- Amortized Analysis (Chapter 11)
- Result: *M* merges take time *M* log *n*

  ⇒ amortized complexity of all ops =

Comparing Priority Queues

• Binary Heaps	• Leftist Heaps
• d-Heaps	• Skew Heaps

Student Activity