CS 261

Skew Heaps

Tim Budd
Sinisa Todorovic
Same idea, different technique

• The same heap order property, but no complete tree requirement

• Order of left and right children is unimportant

• Addition and remove are special cases of merge
Removal as Merge

• When you remove the root, you are left with two child trees
• Merge to form the new heap

```java
void removeFirst() {
    assert (root != 0);
    root = merge(root.left, root.right);
}
```
But addition as merge?

- Addition is the merge of
  - The existing heap, and
  - A new heap that has only one element (the element being added).

```c
void add (TYPE newValue) {
    root = merge(root, newNode(newValue));
}
```
There must be a trick

To merge, take smaller of the two
Then swap the children, and recursively merge.

The swapping is key, if things get unbalanced, it keeps them from staying so
Merge algorithm

Node * merge (Node *left, Node *right)
    if (left == NULL) return right;
    if (right == NULL) return left;
    if (left->value < right->value) {
        small = left;  big = right;
    } else {
        small = right;  big = left;
    }
    Node * temp = small->right;
    small->right = small->left;
    small->left = merge(temp, big)
    return small;
}
Example, merge two trees
Next Step
Why Long Trees cannot stay so
Example
In practice?

- Amortized $O(\log n)$
- Not guaranteed $O(\log n)$ as in the heap
- But in practice just as fast