CSCI 136
Data Structures &
Advanced Programming

SkewHeaps

Video Outline

- Skew Heaps
  - Why
  - What
  - How

Merging Heaps

- Goal: We want to build a *very large* heap.
- Suppose we have a huge data set.
  - We'd like partition the data, build smaller heaps in parallel, and then merge them together

- How long to merge two `VectorHeaps`?
- How complicated is it?

- Is a `VectorHeap` the right tool for the job?

Revisiting Heap Design

- Think back to the trade-offs between vectors and lists
  - Inserting into a vector requires shifting, but inserting into a linked structure can be done by updating references

- Observation: Heaps don't *need* to be array-based complete binary trees. An arbitrary binary tree can satisfy the heap invariants too

BinaryTree-based Heaps

- Downsides to using `BinaryTree` objects to store our heaps?
  - We waste a small amount of space per node
    - Each `BinaryTree` node holds three extra references
  - We don't guarantee balance
    - This may be a trade-off we are OK with as long as things don't get *too* bad…
- Upsides?
  - Updating references is fast (no copying/shifting arrays): this may open doors to new functionality

Mergeable Heaps

- Consider the *destructive* operation:

```
merge(heap1, heap2)
```

- Implementing heap operations become relatively straightforward with merge as a building block!
  - Get: return the value stored in the root
  - Add: merge with the single-element heap
  - Remove: detach the root from its subtrees, then merge the old left and right subtrees

Mergeable Heaps

Implementing `merge(heap1, heap2)`

- Basic idea: the heap with highest priority root somehow "absorbs" the heap with lower-priority root as a subtree
- Challenges:
  - "Absorbs" how? Where?
  - How much reheapifying is needed
  - How deep do trees get after many merges?

Skew Heap: Merge Pseudocode

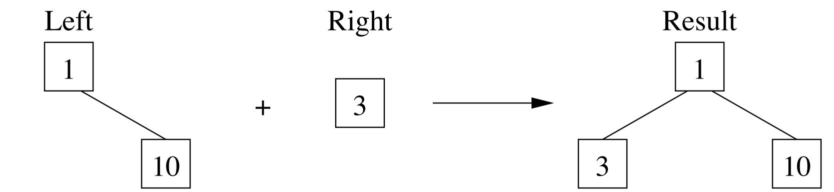
```
SkewHeap merge(SkewHeap L, SkewHeap R)
    if either L or R is empty:
        return the other

    if L.minValue < R.minValue:
        swap L and R   (now L has minValue)

    if L has no left subtree:
        set R as L's left subtree

    else:
        swap L's left and right subtrees
        let temp ← L's left subtree
        set L's left child ← merge(temp, R)
```

Case 1

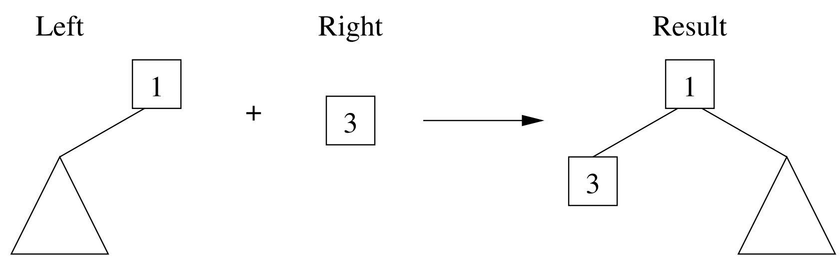
Case 2

Case 3 (recurse)

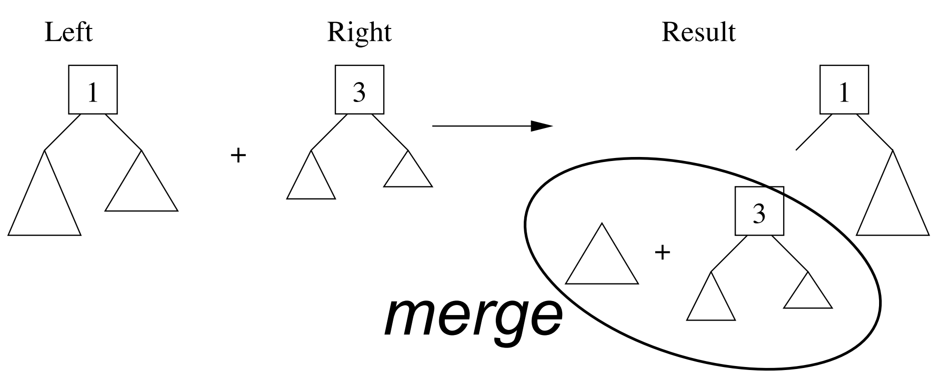
Skew Heap: Merge Examples



Case 2



Case 3 (recurse)



Bailey
page 331

Skew Heap Performance

- Code & low-level details are in the textbook, but at a high level…
  - The merge algorithm makes no guarantees for any individual operation, but it keeps the tree shallow over time—the amortized behavior is good
  - Theorem: Any set of `m` SkewHeap operations can be performed in `O(m log n)` time, where `n` is the total number of items in the SkewHeaps

Heap Summary

- Heaps are a partially ordered tree based on item priority
  - Invariants: parent has higher priority than each child
- Heaps provide:
  - and efficient `PriorityQueue` implementation
  - an efficient building block for sorting (heapsort)
- We can efficiently manage heaps in an implicit array representation
- But we can add flexibility and functionality if we carefully manage heaps using binary trees