Leftist Heaps and Skew Heaps

Leftist Heaps

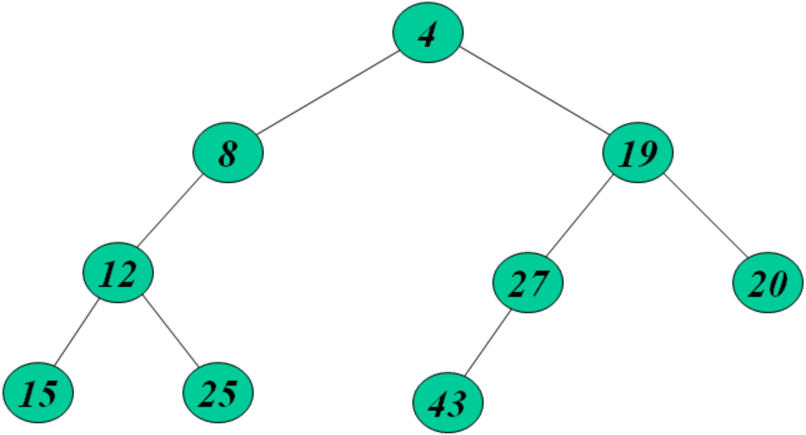
- A binary heap provides $O(\log n)$ inserts and $O(\log n)$ deletes but suffers from $O(n \log n)$ merges
- A leftist heap offers $O(\log n)$ inserts and $O(\log n)$ deletes *and* $O(logn)$ merges
- Note, however, leftist heap inserts and deletes are *more expensive* than Binary Heap inserts and deletes

Leftist Heaps: Definition

- A *Leftist (min)Heap* is a binary tree that satisfies the following conditions. If X is a node and L and R are its left and right children, then:
  1. X.value $\leq$ L.value
  2. X.value $\leq$ R.value
  3. null path length of L $\geq$ null path length of R

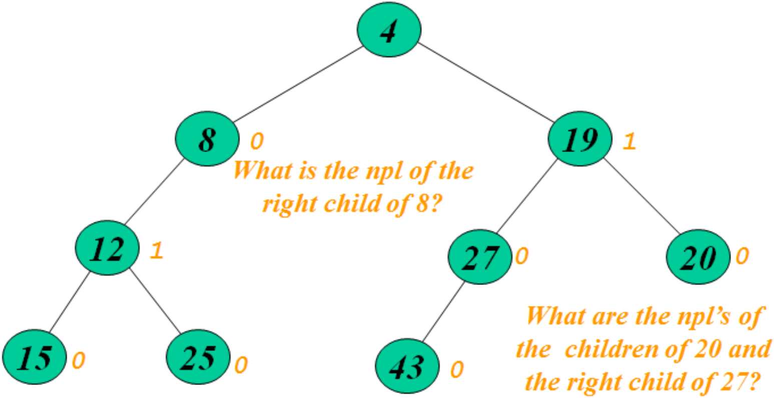
  where the *null path length* of a node is the shortest between from that node to a descendant with 0 or 1 child. If a node is null, its null path length is -1.

Example: Null Path Length



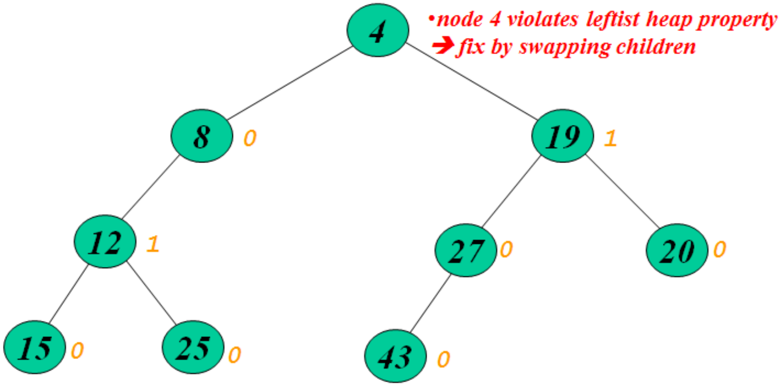
node	4	8	19	12	15	25	27	20	43
npl	1	0	1	1	0	0	0	0	0

Example: Null Path Length



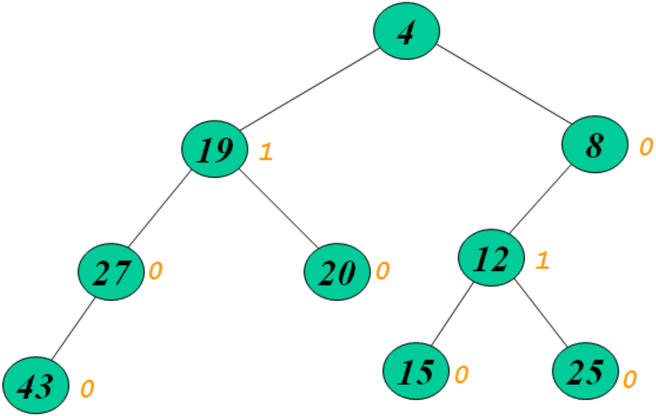
node	4	8	19	12	15	25	27	20	43
npl	1	0	1	1	0	0	0	0	0

Example: Null Path Length



node	4	8	19	12	15	25	27	20	43
npl	1	0	1	1	0	0	0	0	0

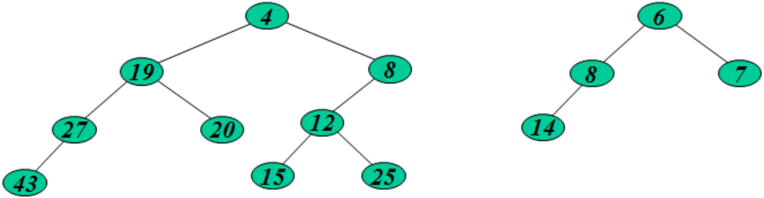
Leftist Heaps



node	4	8	19	12	15	25	27	20	43
npl	1	0	1	1	0	0	0	0	0

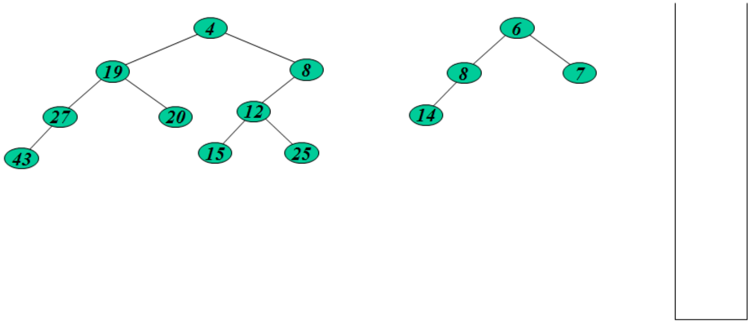
Merging Leftist Heaps

Consider two leftist heaps …



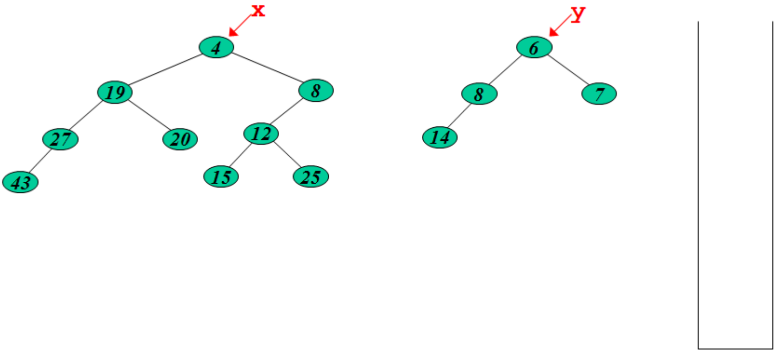
Task: merge them into a single leftist heap

Merging Leftist Heaps



First, instantiate a Stack

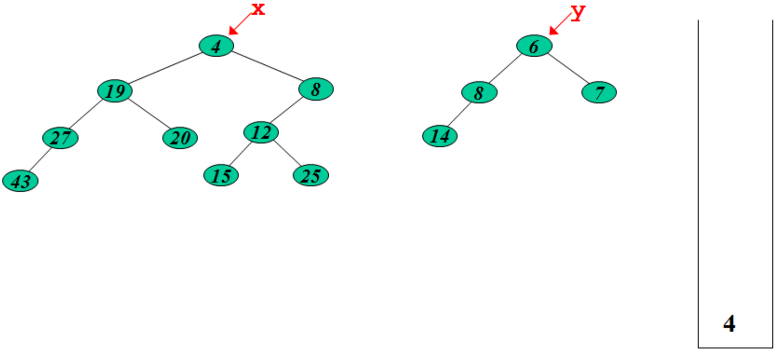
Merging Leftist Heaps



Compare root nodes

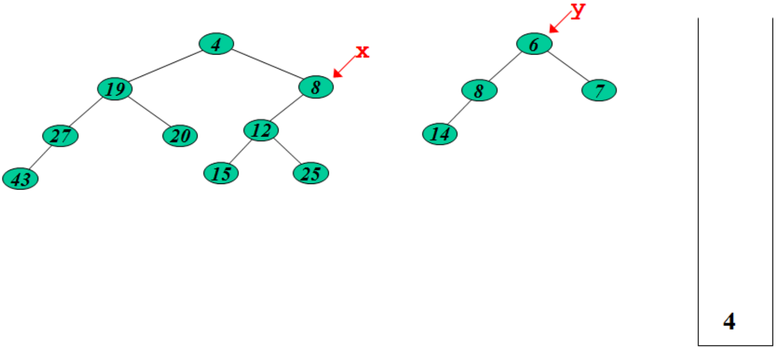
`merge(x,y)`

Merging Leftist Heaps



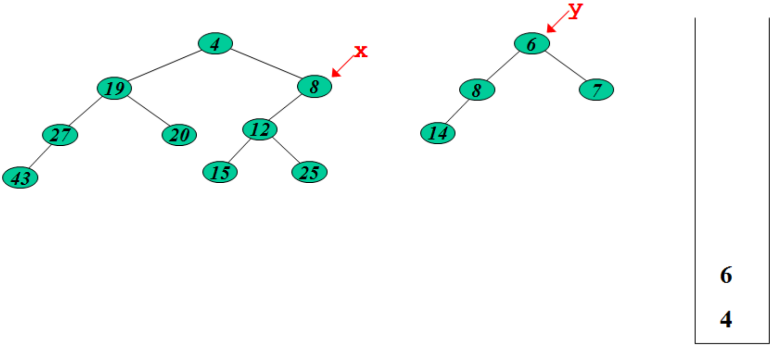
Remember smaller value

Merging Leftist Heaps



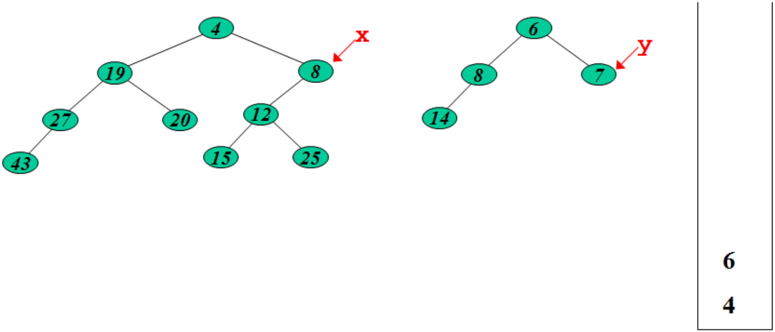
Repeat the process with the right child of the smaller value

Merging Leftist Heaps



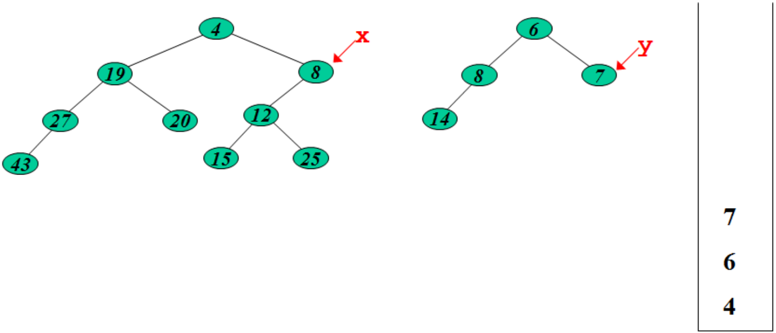
Remember smaller value

Merging Leftist Heaps



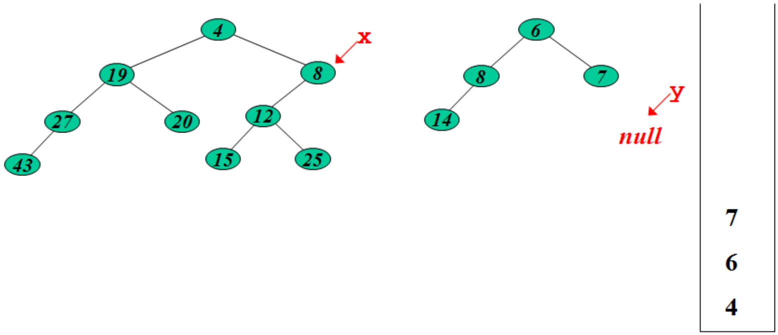
Repeat the process with the right child of the smaller value

Merging Leftist Heaps



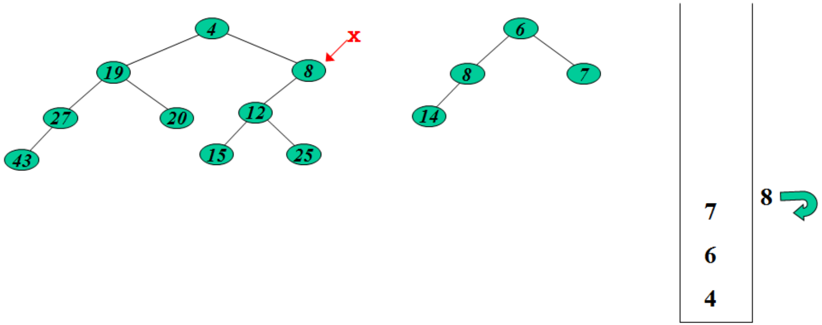
Remember smaller value

Merging Leftist Heaps



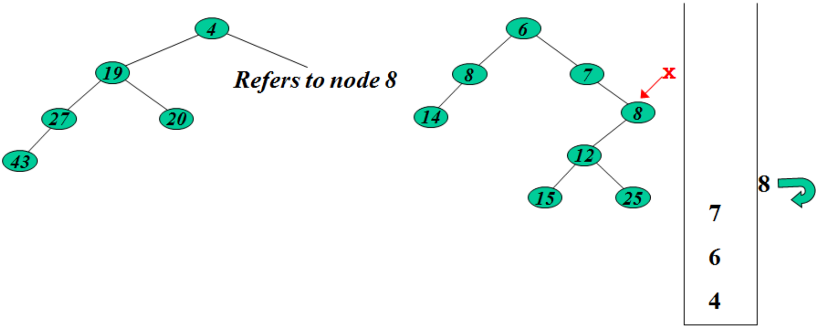
Repeat the process with the right child of the smaller value

Merging Leftist Heaps



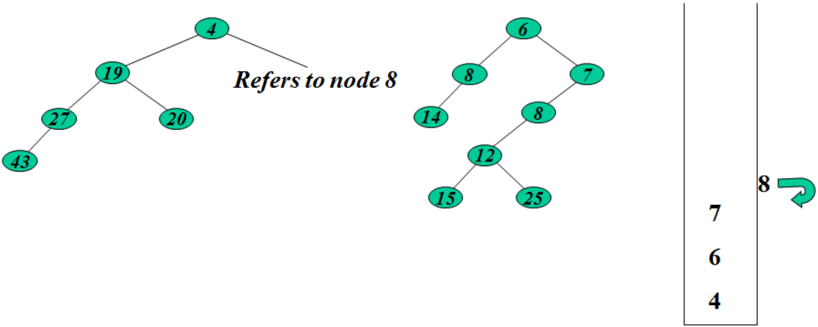
Because one of the arguments is null, return the other argument

Merging Leftist Heaps



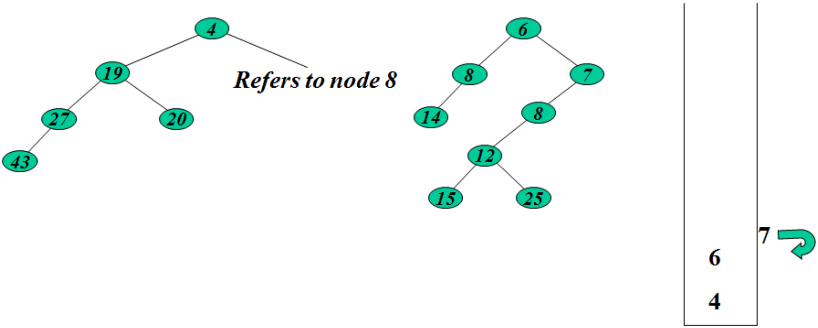
Make 8 the right child of 7

Merging Leftist Heaps



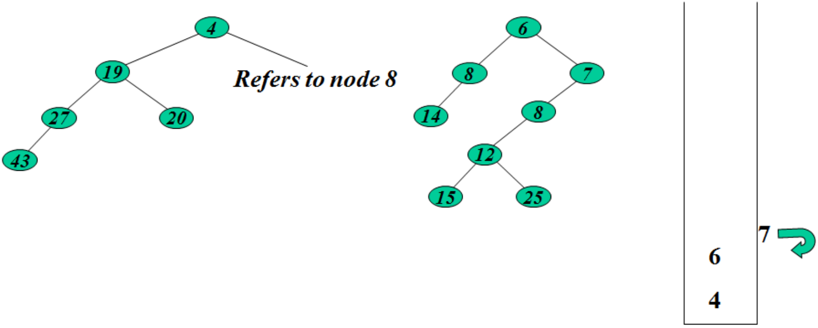
Make 7 leftist (by swapping children)

Merging Leftist Heaps



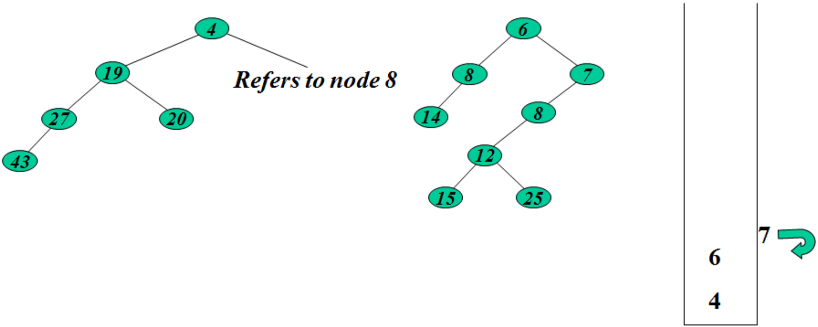
Return node 7

Merging Leftist Heaps



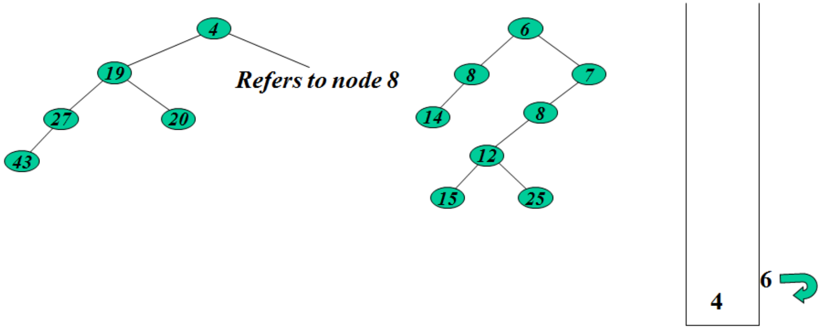
Make 7 the right child of 6 (which it already is)

Merging Leftist Heaps



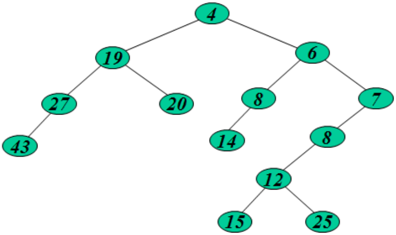
Make 6 leftist (it already is)

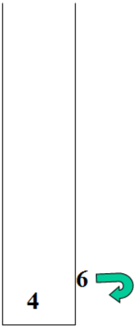
Merging Leftist Heaps



Return node 6

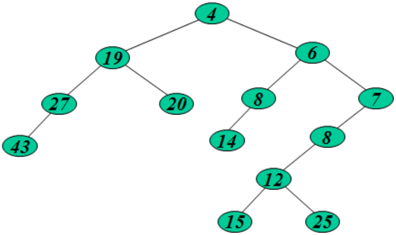
Merging Leftist Heaps





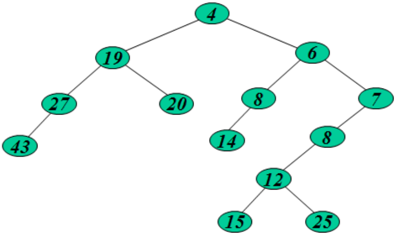
Make 6 the right child of 4

Merging Leftist Heaps



Make 4 leftist (it already is)

Final Leftist Heap



- Verify that the tree is heap
- Verify that the heap is leftist

Return node 4

Analysis

- Height of a leftist heap $\approx O(\log n)$
- Maximum number of values stored in Stack $\approx 2 * O(\log n) \approx O(\log n)$
- Total cost of merge $\approx O(\log n)$

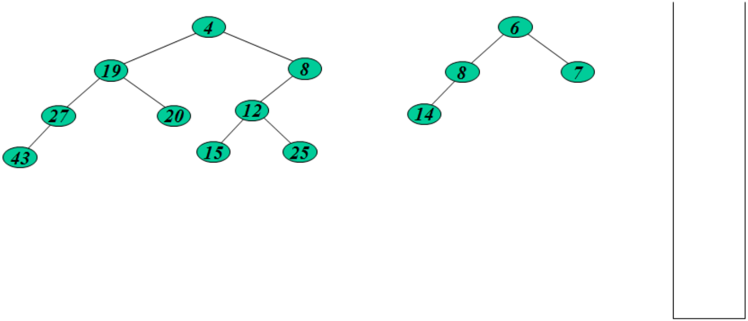
Insert and Delete

- To insert a node into a leftist heap, merge the leftist heap with the node
- After deleting root X from a leftist heap, merge its left and right subheaps
- In summary, there is only one operation, a merge.

Skew Heaps

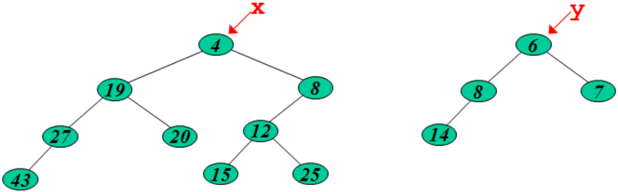
- Simplify leftist heap by
  - not maintaining null path lengths
  - swapping children at every step
- A *Skew (min)Heap* is a binary tree that satisfies the following conditions. If X is a node and L and R are its left and right children, then:
  1. X.value $\leq$ L.value
  2. X.value $\leq$ R.value
- A *Skew (max)Heap* is a binary tree that satisfies the following conditions. If X is a node and L and R are its left and right children, then:
  1. X.value $\geq$ L.value
  2. X.value $\geq$ R.value

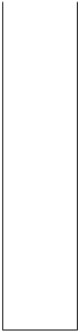
Merging Skew Heaps



First, instantiate a Stack

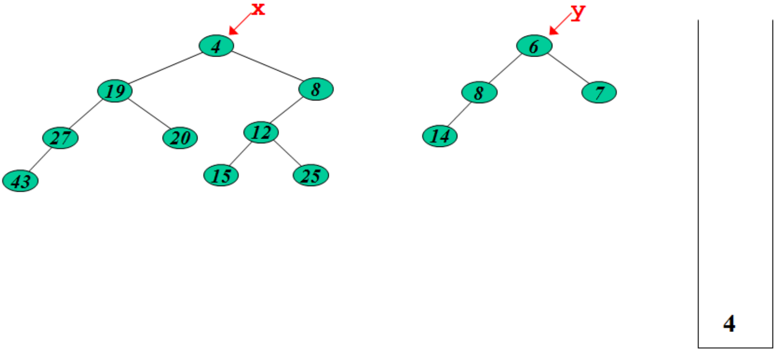
Merging Skew Heaps





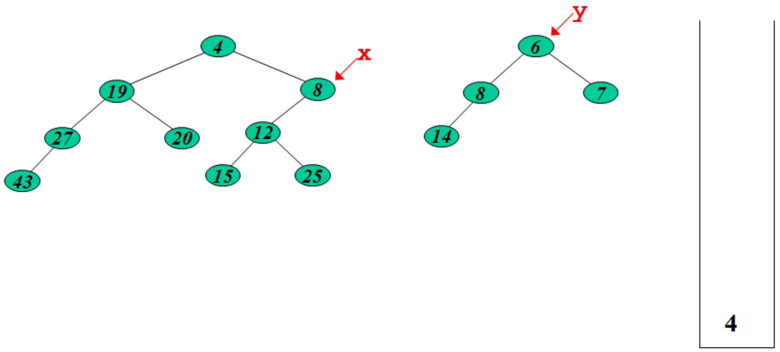
Compare root nodes
`merge(x,y)`

Merging Skew Heaps



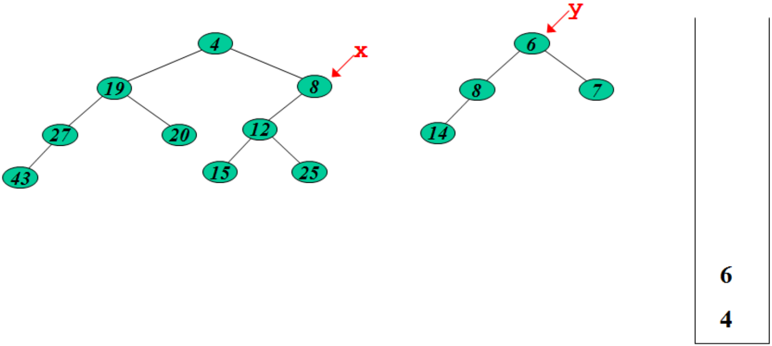
Remember smaller value

Merging Skew Heaps



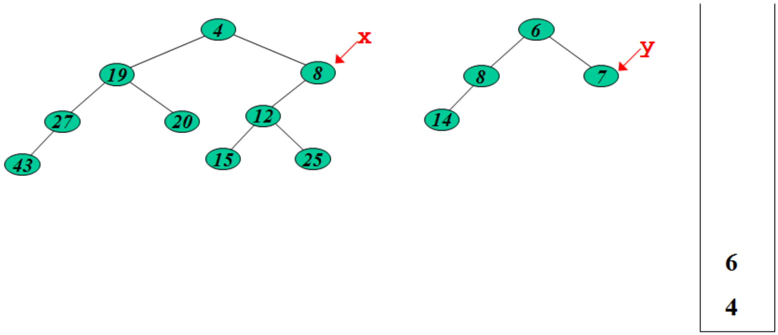
Repeat the process with the right child of the smaller value

Merging Skew Heaps



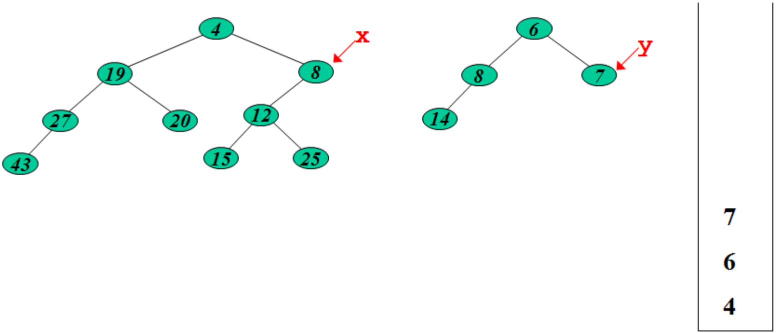
Remember smaller value

Merging Skew Heaps

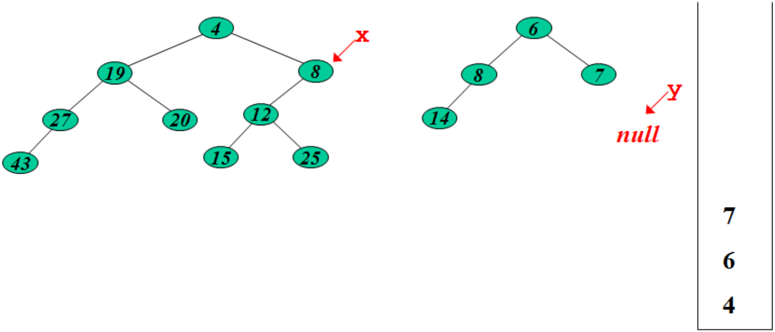


Repeat the process with the right child of the smaller value

Merging Skew Heaps

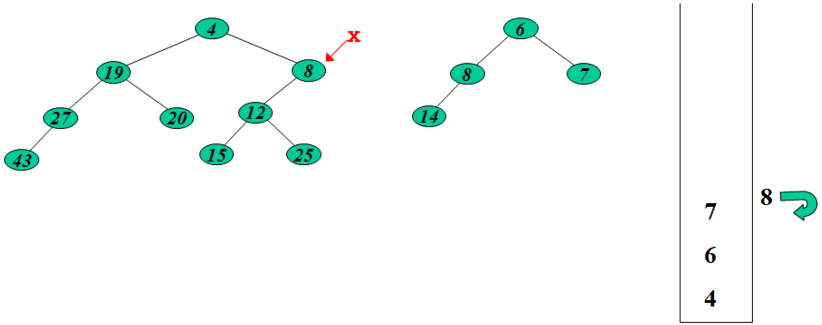


Remember smaller value



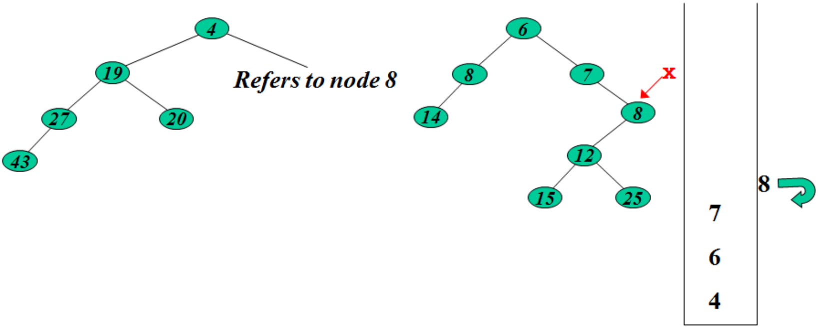
Repeat the process with the right child of the smaller value

Merging Skew Heaps



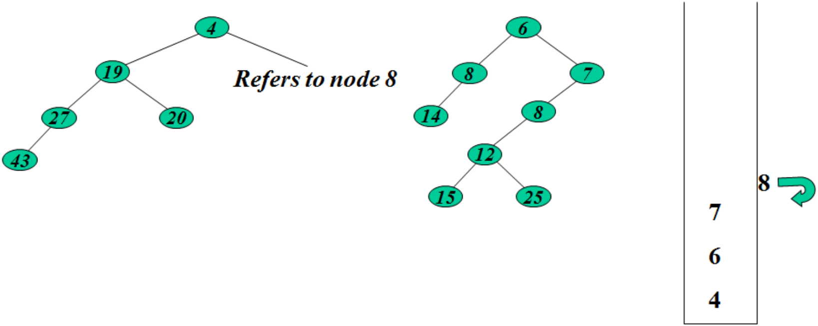
Because one of the arguments is null, return the other argument

Merging Skew Heaps



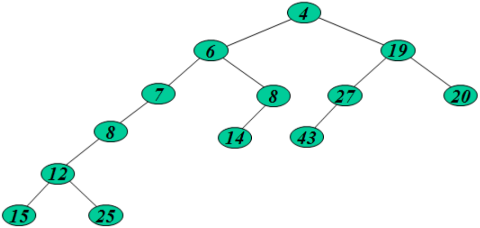
Make 8 the right child of 7

Merging Skew Heaps

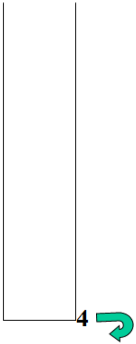


Swap children of node 7

Final Skew Heap



- **Verify that the tree is heap**
- **Verify that the heap is skew**



Return node 4