# Honors Data Structures

Lecture 17: More Priority Queues: DecreaseKey; Leftist Heaps

3/28/22

Daniel Bauer

# DecreaseKey

- What do we do if the priority of an entry in the heap changes?

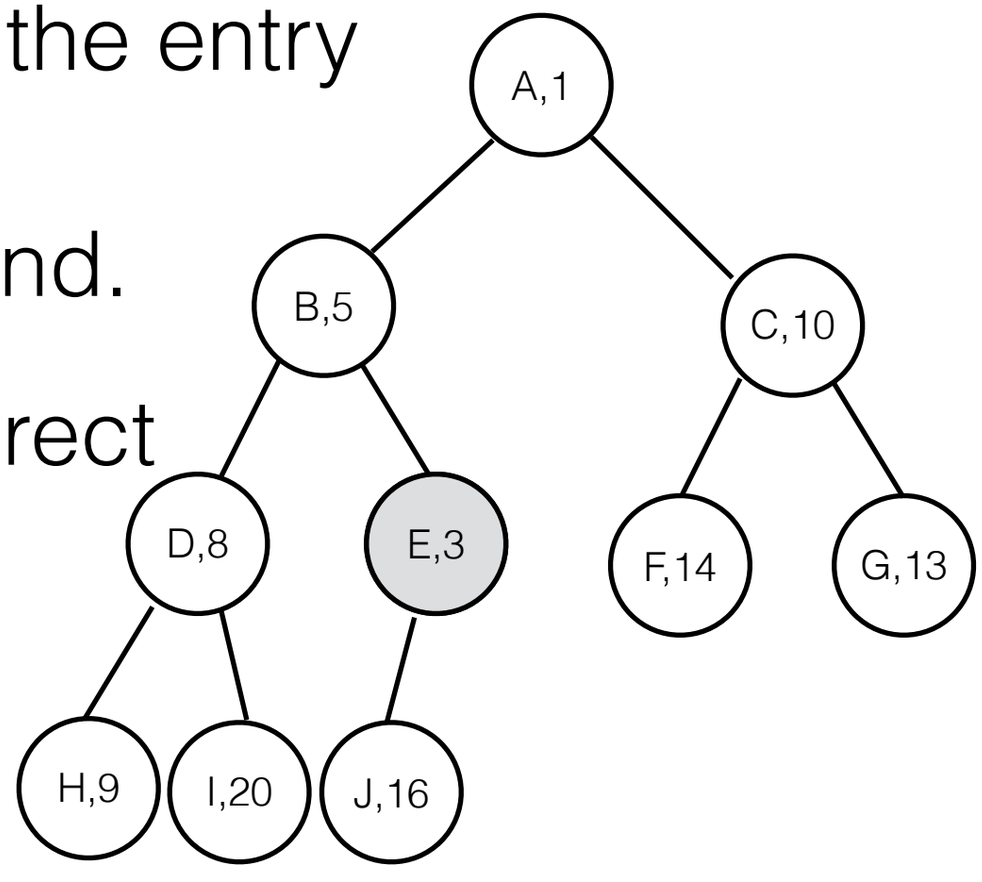
decreaseKey(E, 3)

- Step 1: Find the original copy of the entry in the heap.

  Use hash map for O(1) find.

- Step 2: Move that entry to its correct position.

  Use percolateUp.




| | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | A,1 | B,5 | C,10 | D,8 | E,3 | F,14 | G,13 | H,9 | I,20 | J,16 | | | |

# DecreaseKey

- What do we do if the priority of an entry in the heap changes?

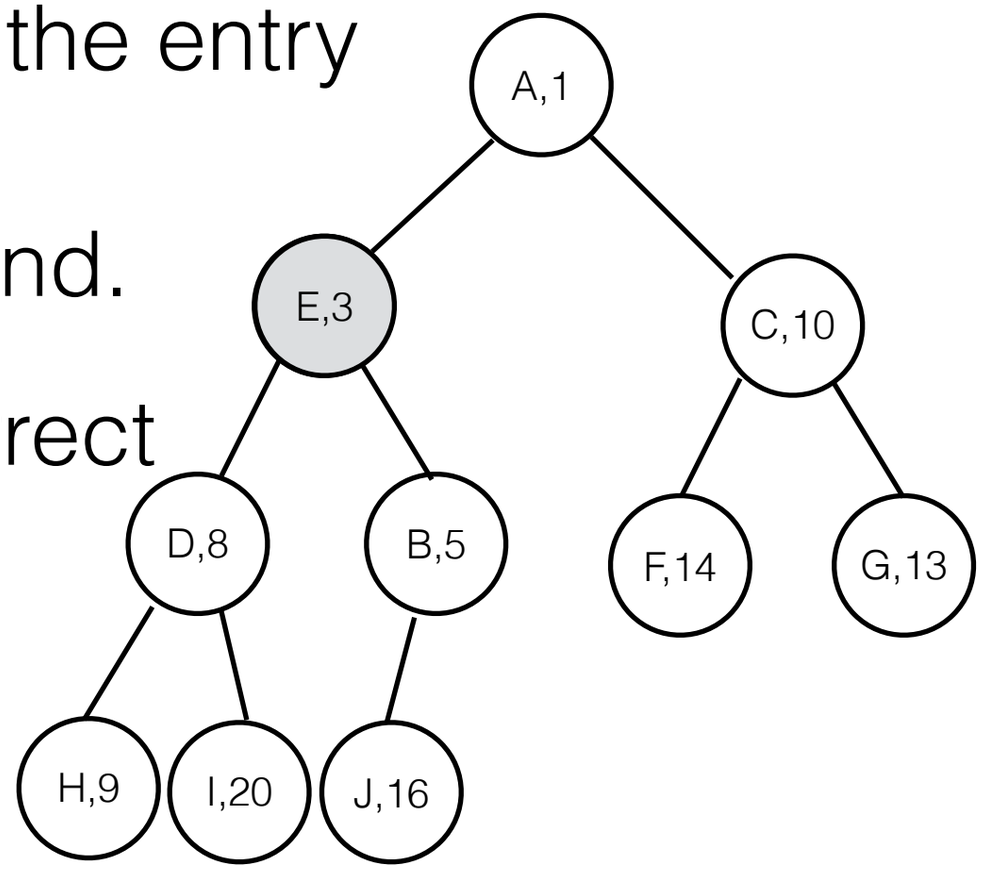
  decreaseKey(E, 3)

- Step 1: Find the original copy of the entry in the heap.

  Use hash map for O(1) find.

- Step 2: Move that entry to its correct position.

  Use percolateUp.



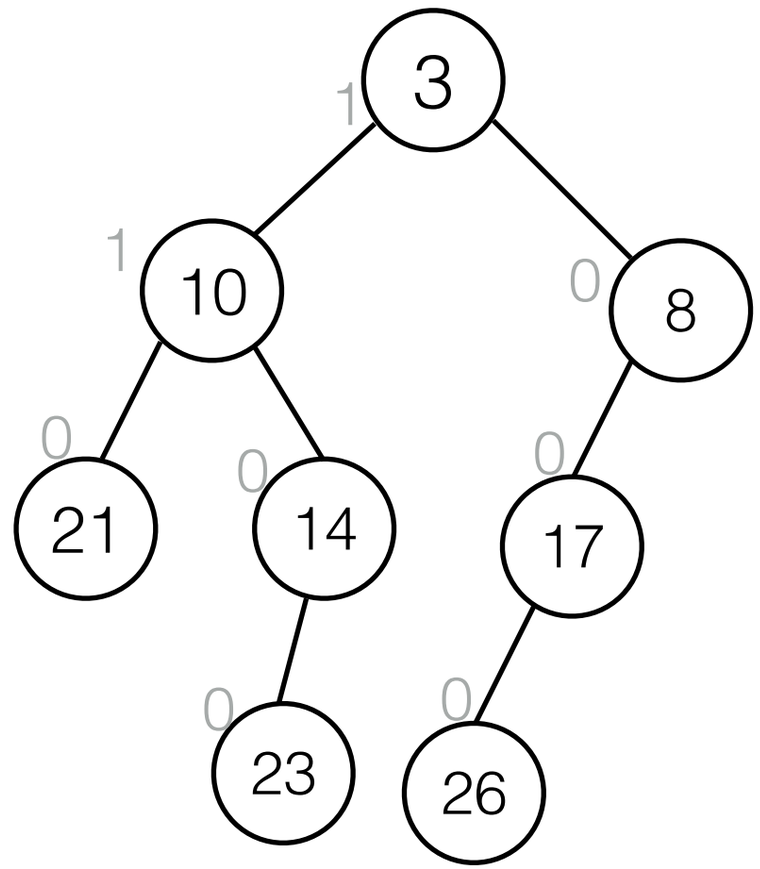

| | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | A,1 | E,3 | C,10 | D,8 | B, 5 | F,14 | G,13 | H,9 | I,20 | J,16 | | | |

# Leftist Heaps

- How can we implement a heap using immutable data structures?
  - Cannot use an array, so use regular binary tree implementation (linked nodes).
- Basic idea: A data structure that allows you to efficiently **merge** two heaps.
  - `insert:` create a new heap with a single node, merge with the existing heap.
  - `deleteMin:` remove the root node, merge the two subtrees.

# Null-path length

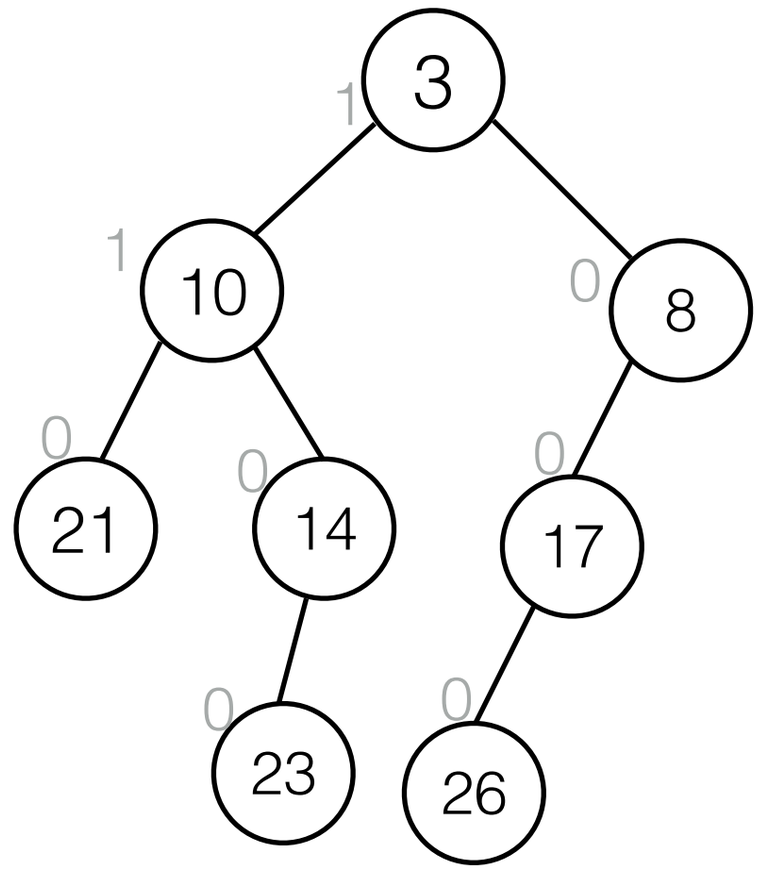
- The **null-path length** of a node *n*, is the length of the shortest path from *n* to a node with 0 or 1 children. (distance to the closest "null" node)



# Leftists Trees

- In a **leftist tree**, for each node, the null-path length of its left subtree is greater than or equal the null-path length of its right subtree.

- a **leftist heap** is a leftist tree that obeys the heap-order property.

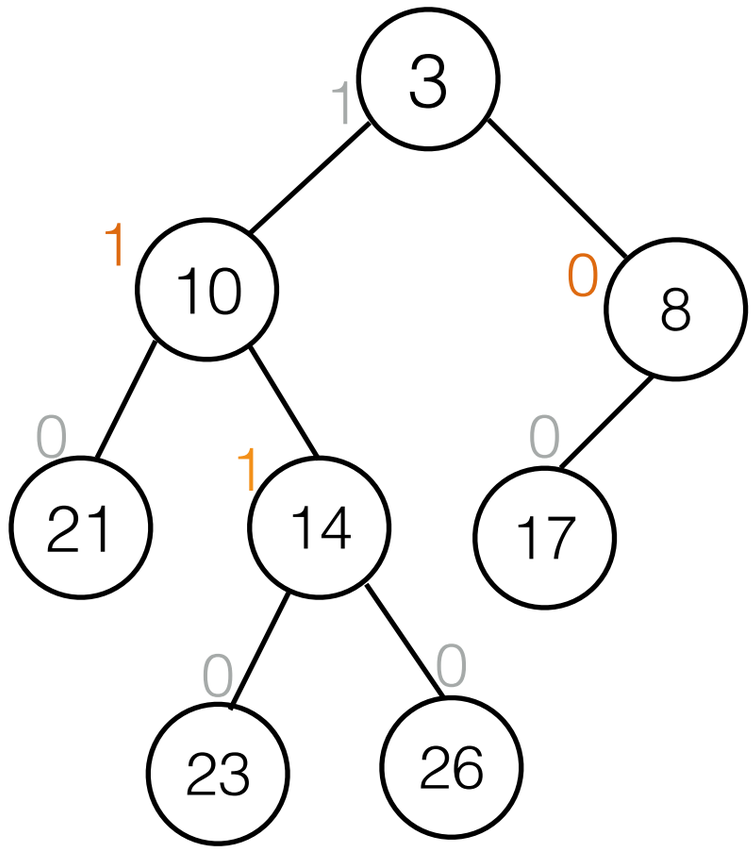



leftist tree

# Leftists Trees

- In a **leftist tree**, for each node, the null-path length of its left subtree is greater than or equal the null-path length of its right subtree.

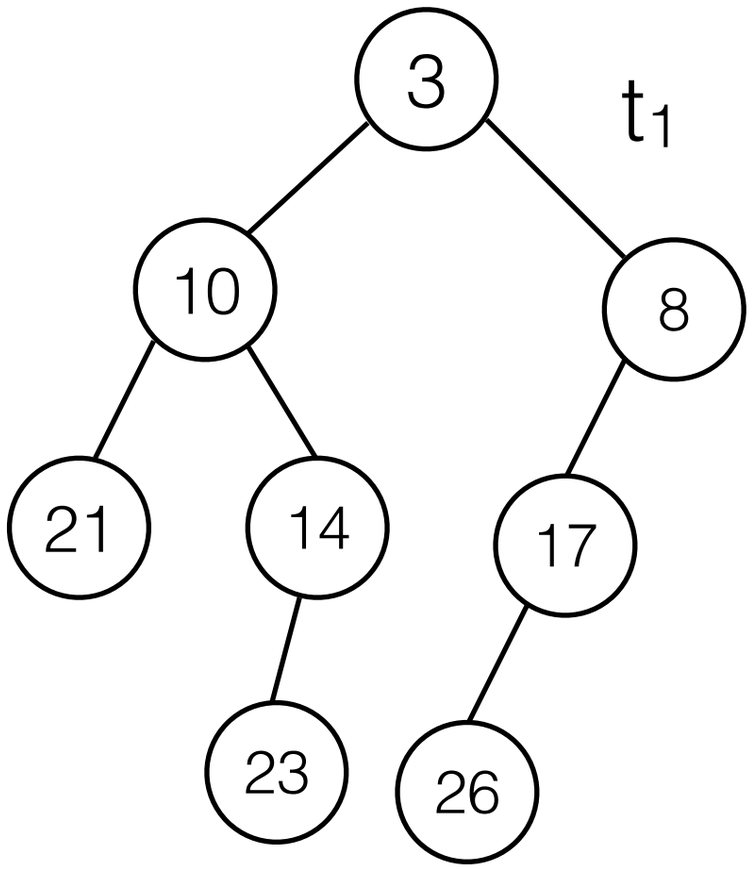
- a **leftist heap** is a leftist tree that obeys the heap-order property.

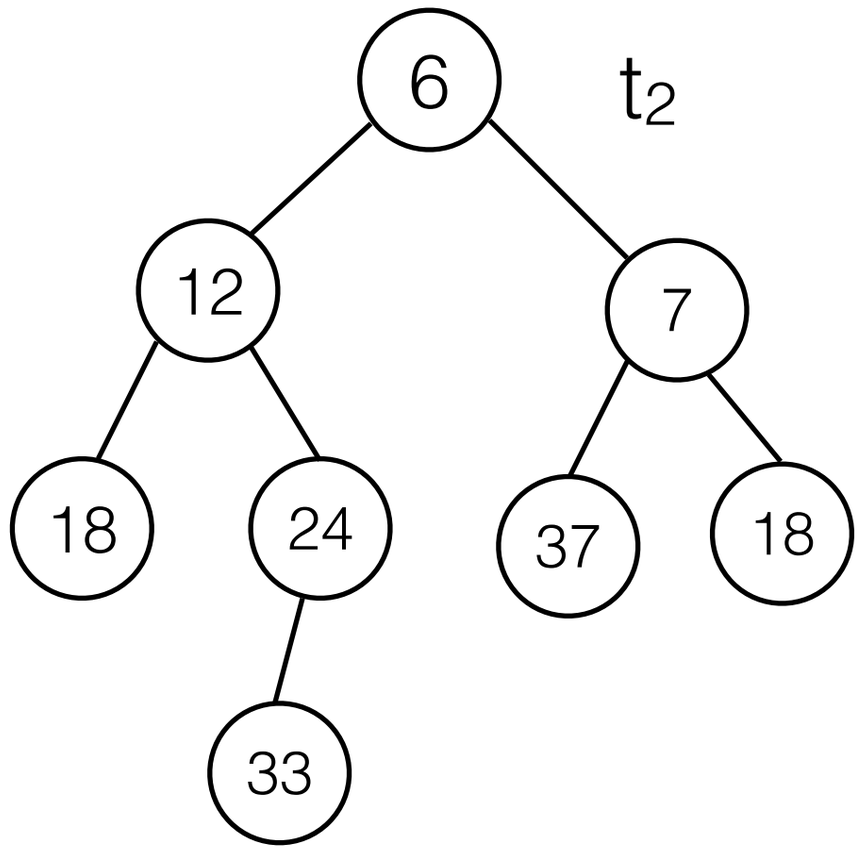



not a leftist tree

# Merging leftist heaps

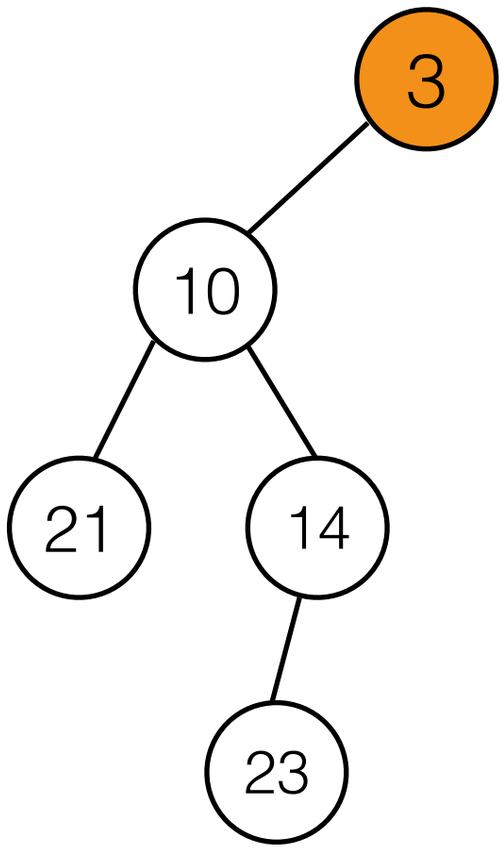
- Compare the two root nodes. Let $t_1$ be the tree with the smaller root and $t_2$ be the tree with the larger root.
  Merge $t_2$ with the right subtree of $t_1$.
- Then attach the result to the right of the root of t1.
- If necessary, flip the subtrees of the smaller root to maintain the leftist property.

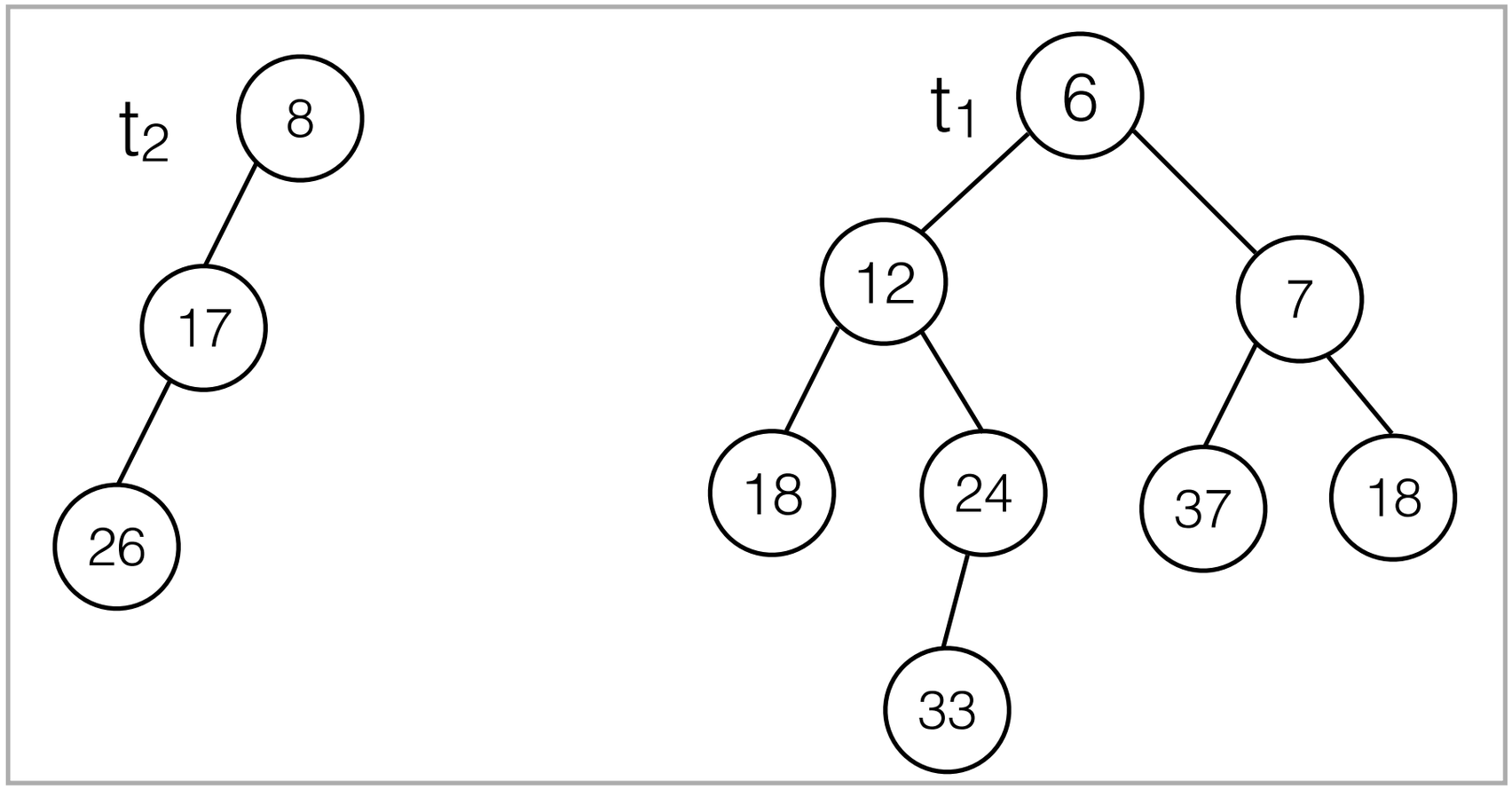


# Merging leftist heaps

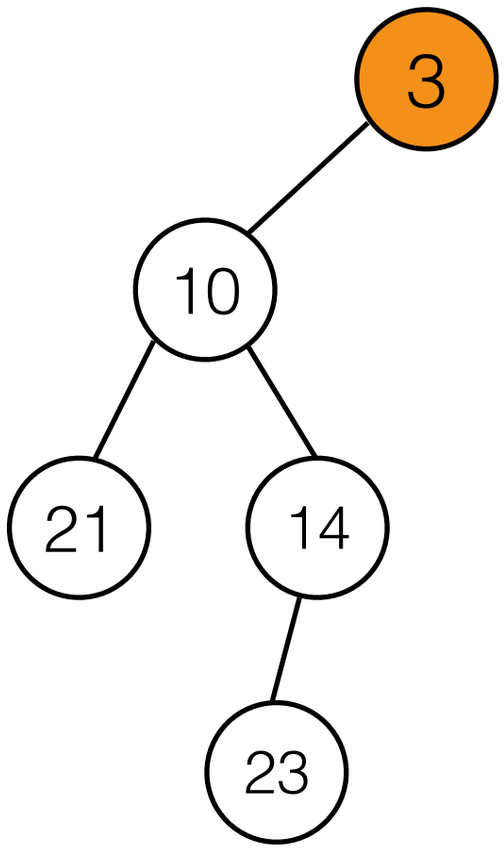
- Compare the two root nodes. Let $t_1$ be the tree with the smaller root and $t_2$ be the tree with the larger root.
  Merge $t_2$ with the right subtree of $t_1$.
- Then attach the result to the right of t1.
- If necessary, flip the subtrees of the smaller root to maintain the leftist property.

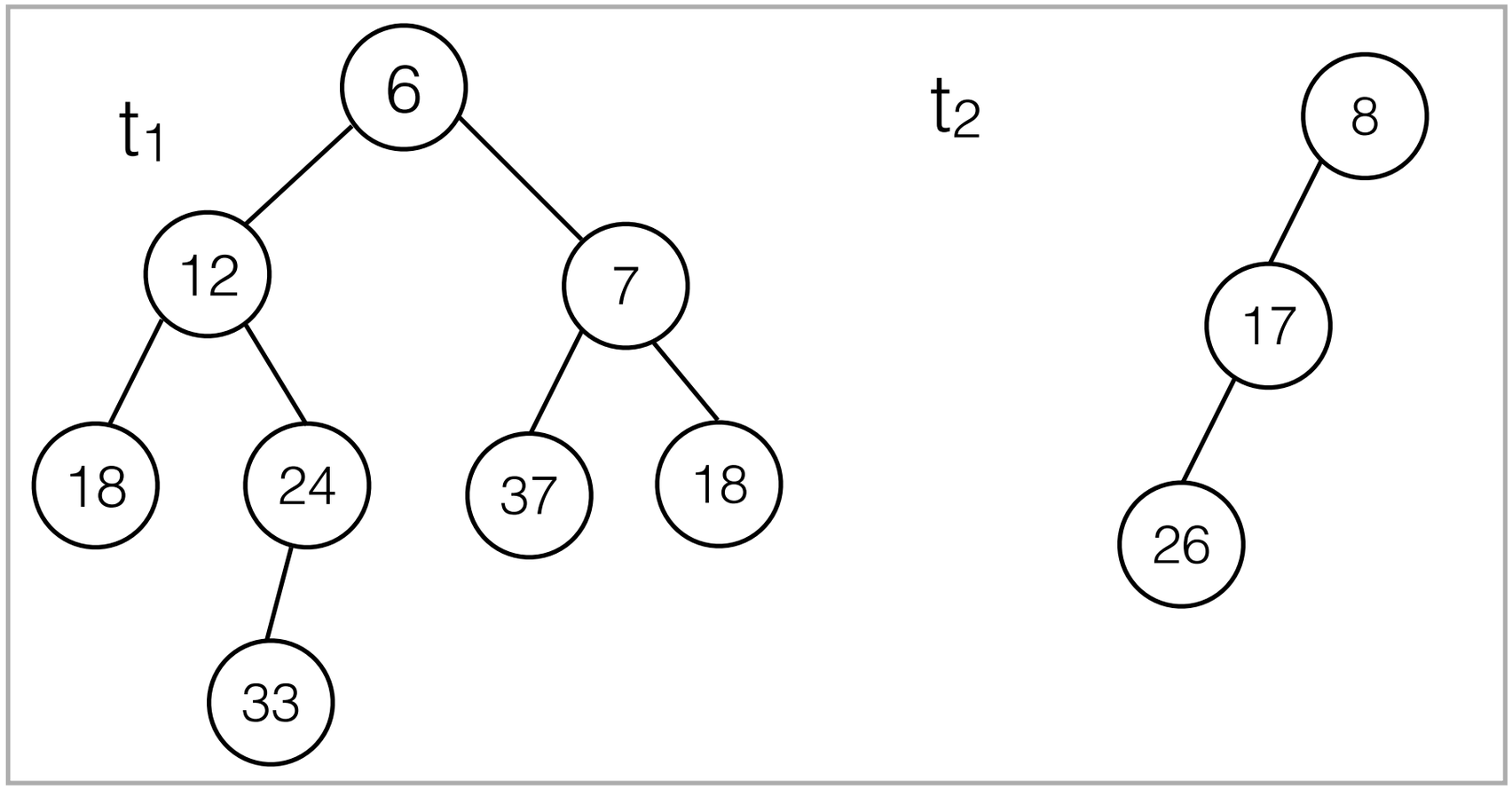


# Merging leftist heaps

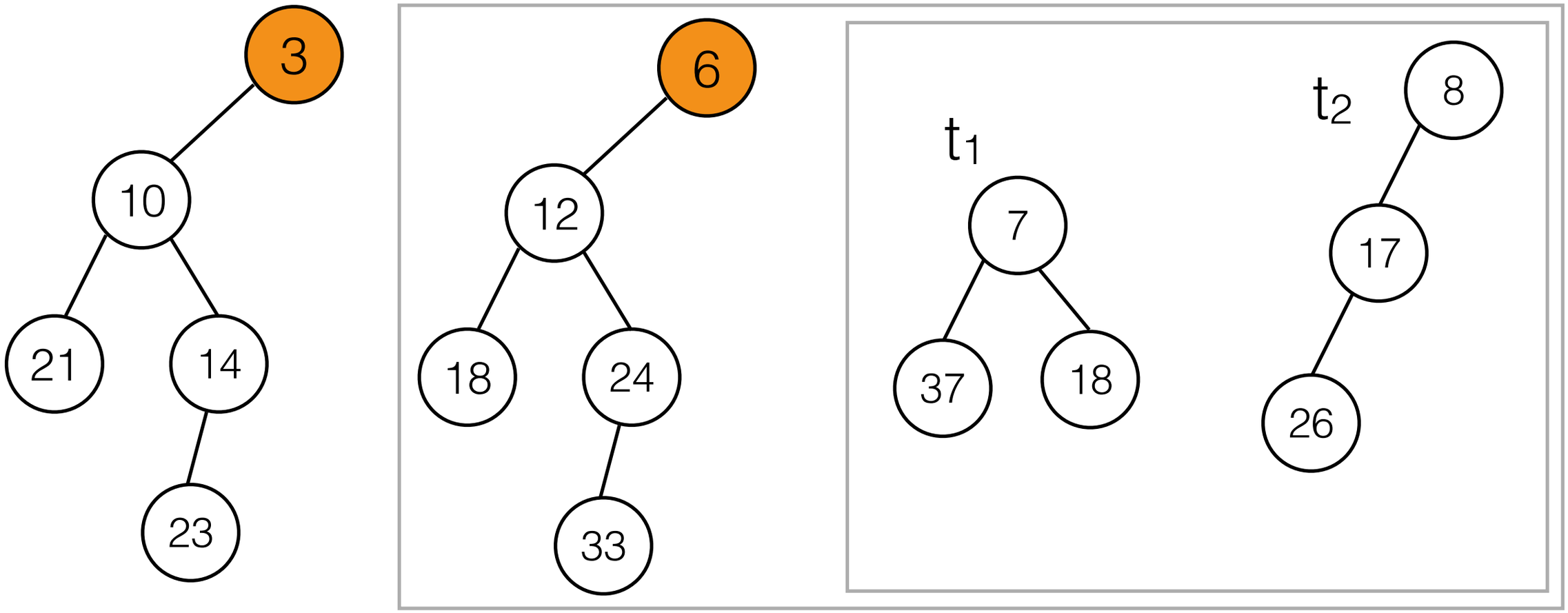
- Compare the two root nodes. Let $t_1$ be the tree with the smaller root and $t_2$ be the tree with the larger root.
  Merge $t_2$ with the right subtree of $t_1$.
- Then attach the result to the right of t1.
- If necessary, flip the subtrees of the smaller root to maintain the leftist property.



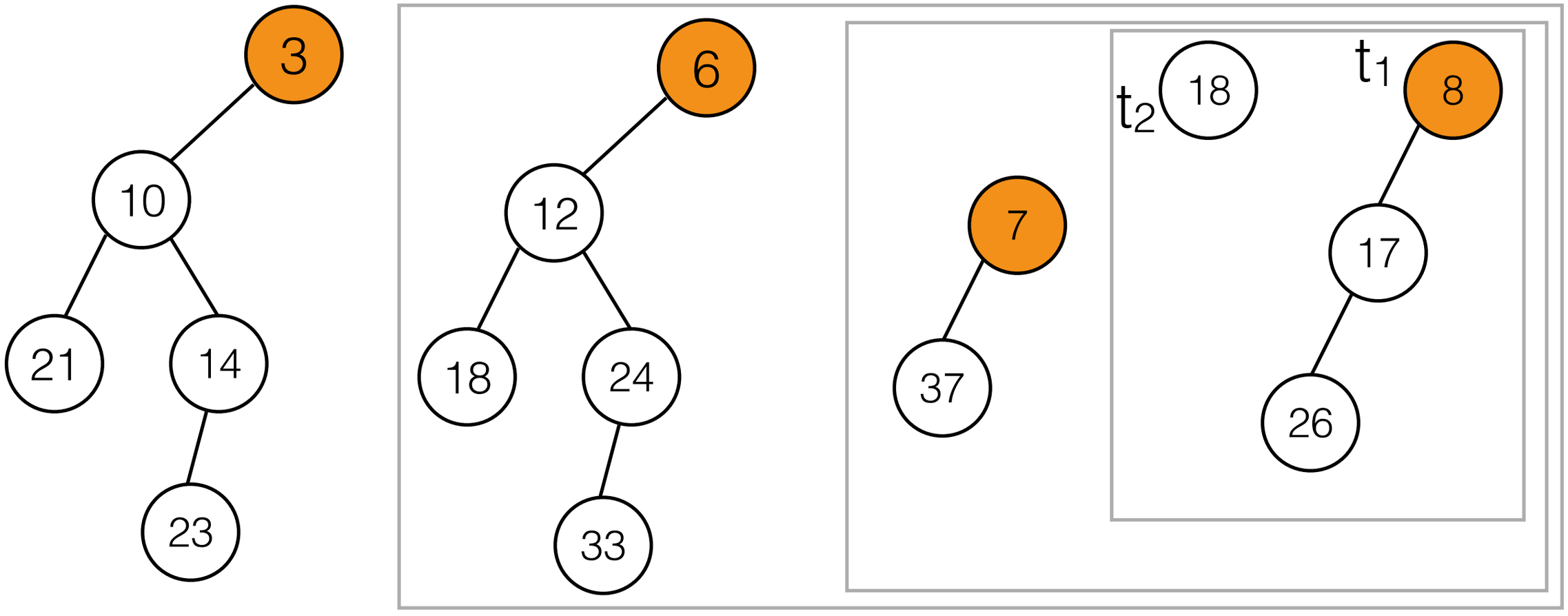
# Merging leftist heaps

- Compare the two root nodes. Let $t_1$ be the tree with the smaller root and $t_2$ be the tree with the larger root.
  Merge $t_2$ with the right subtree of $t_1$.
- Then attach the result to the right of t1.
- If necessary, flip the subtrees of the smaller root to maintain the leftist property.



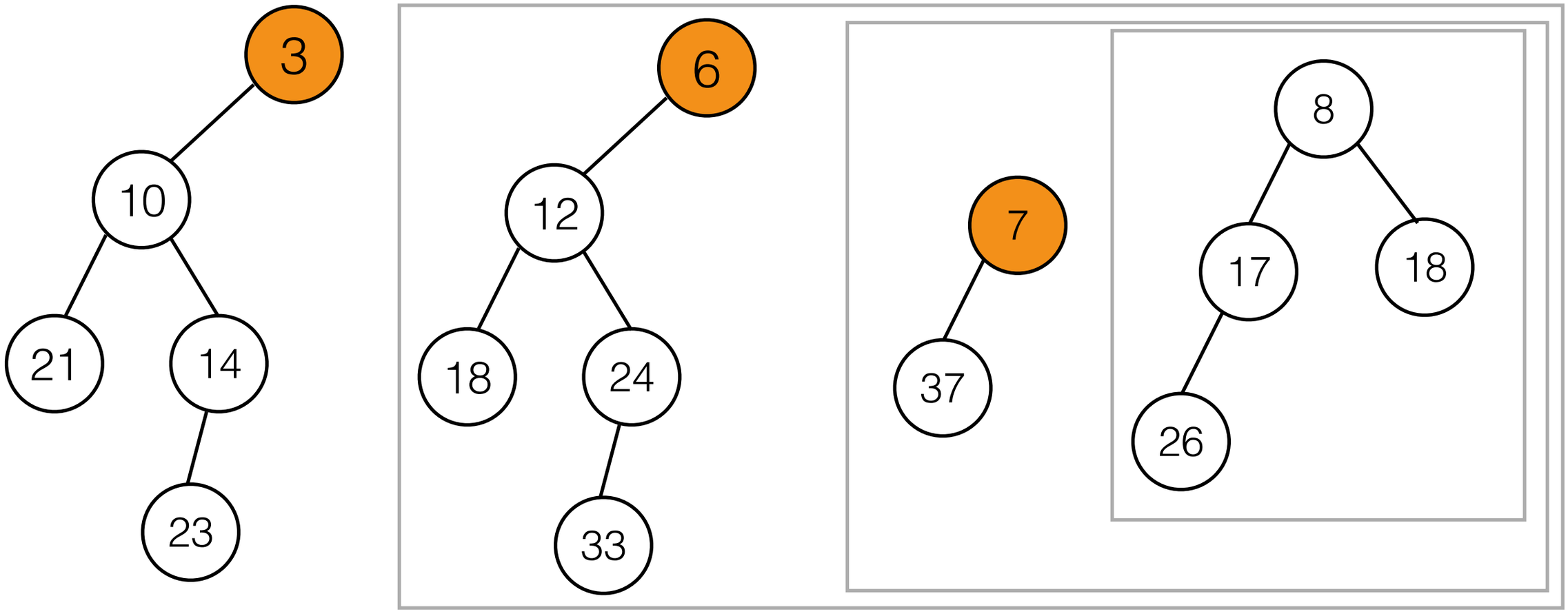
# Merging leftist heaps

- Compare the two root nodes. Recursively merge the tree with the greater root with the right subtree of the tree with the smaller right.
- Make the resulting tree the right subtree of smaller root.
- If necessary, flip the subtrees of the smaller root to maintain the leftist property.



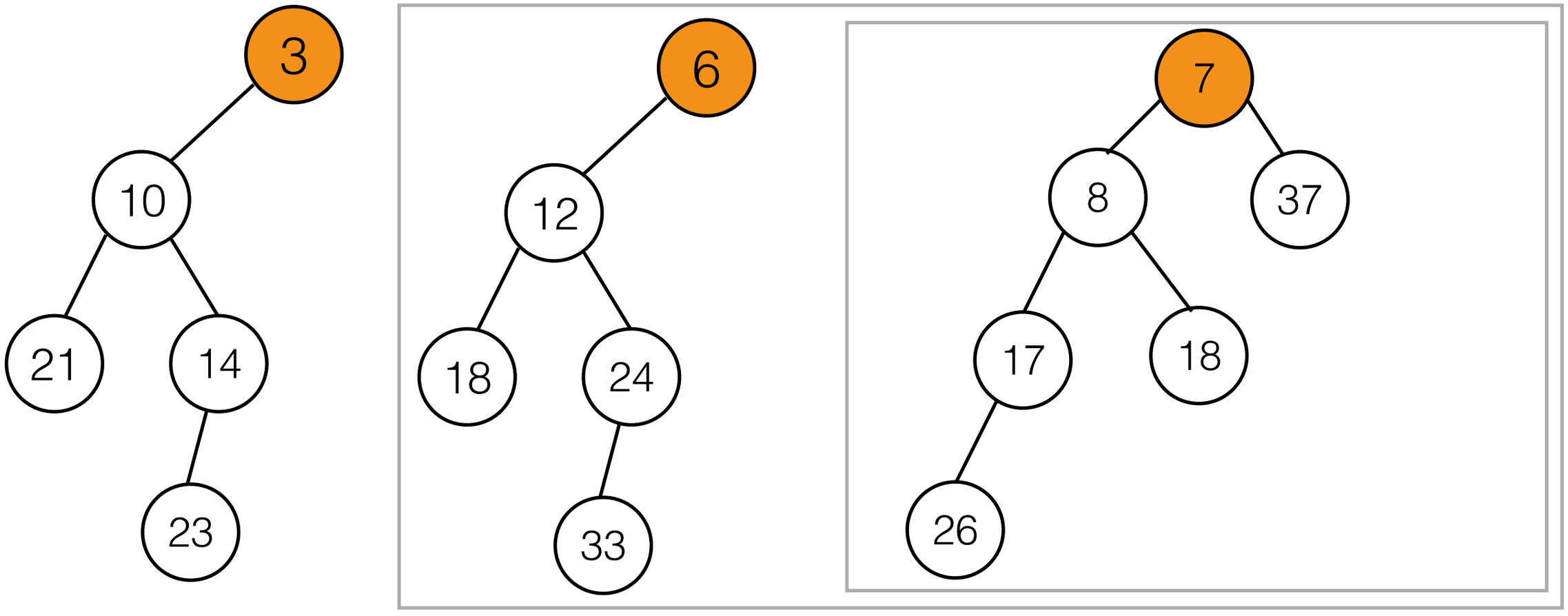
# Merging leftist heaps

- Compare the two root nodes. Recursively merge the tree with the greater root with the right subtree of the tree with the smaller right.
- Make the resulting tree the right subtree of smaller root.
- If necessary, flip the subtrees of the smaller root to maintain the leftist property.



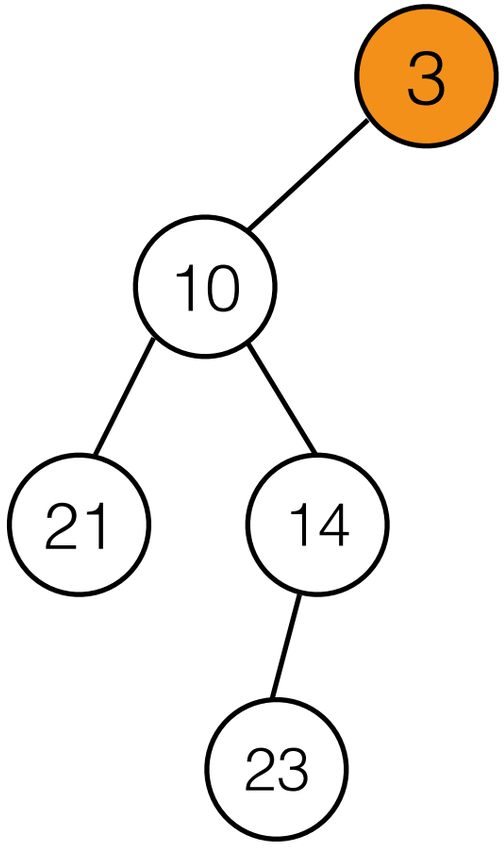
# Merging leftist heaps

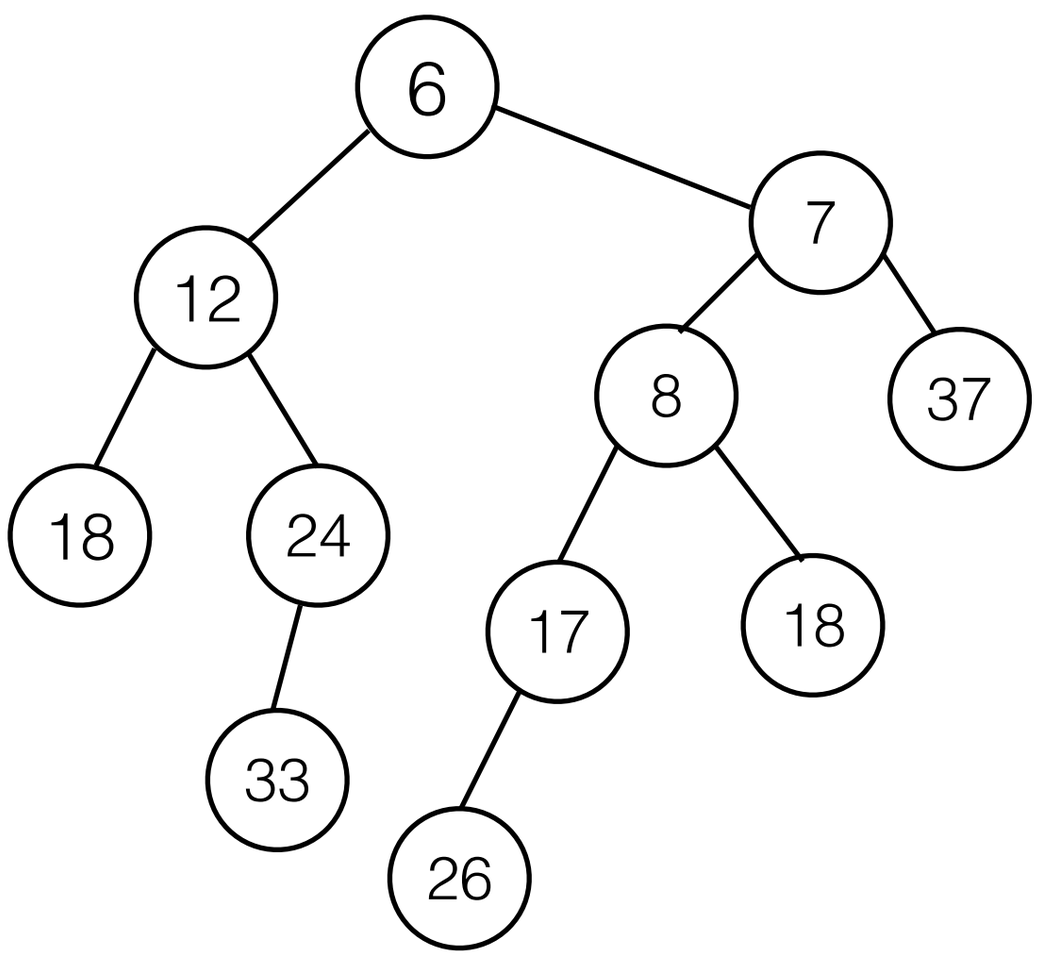
- Compare the two root nodes. Recursively merge the tree with the greater root with the right subtree of the tree with the smaller right.
- Make the resulting tree the right subtree of smaller root.
- If necessary, flip the subtrees of the smaller root to maintain the leftist property.



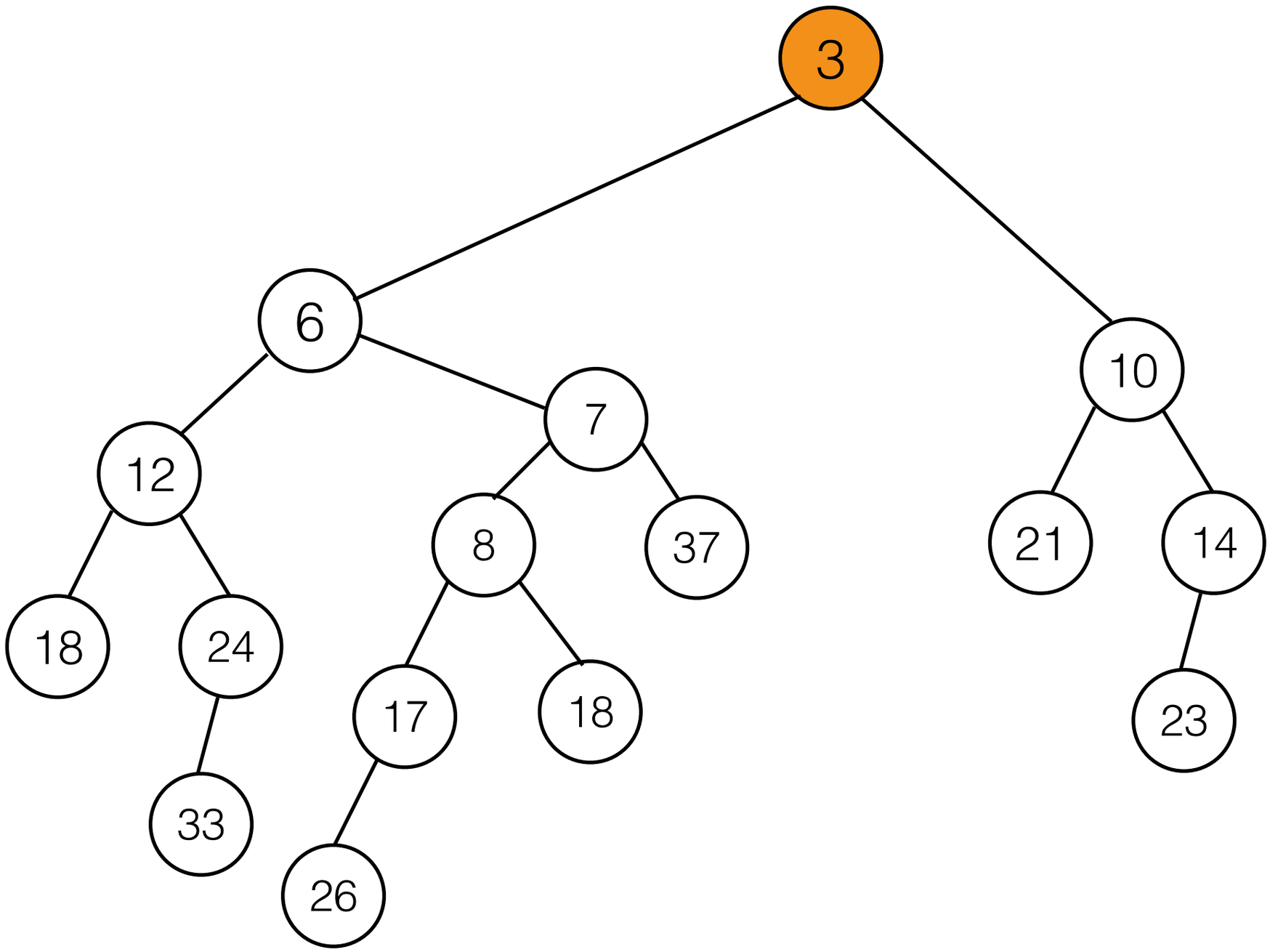
# Merging leftist heaps

- Compare the two root nodes. Recursively merge the tree with the greater root with the right subtree of the tree with the smaller right.
- Make the resulting tree the right subtree of smaller root.
- If necessary, flip the subtrees of the smaller root to maintain the leftist property.



# Merging leftist heaps



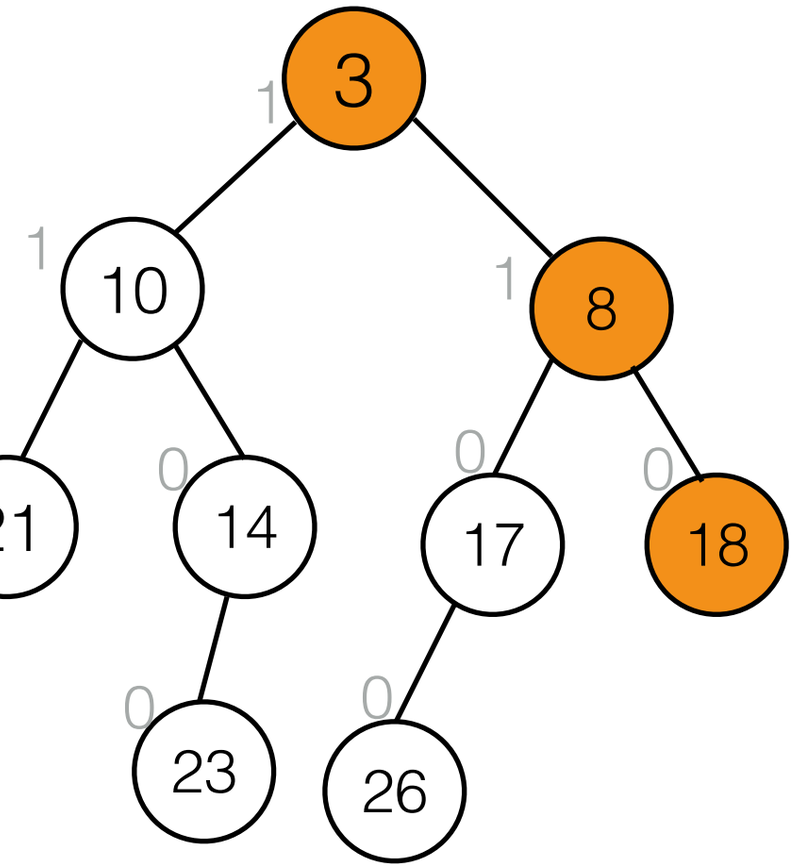
# Length of the Right Path

- The right-(most) path in a leftist tree is as short as any path  from the root to a leaf!

- In a leftist tree with N nodes, right path has at most log2(N+1) nodes.

# Length of the Right Path

- In a leftist tree with N nodes, the null path length at the root node is at most
  ⌊log(N+1)⌋ nodes.




- Therefore, the merge operation takes at most
  O(log (M+N)), where M and N are the sizes of the two trees.