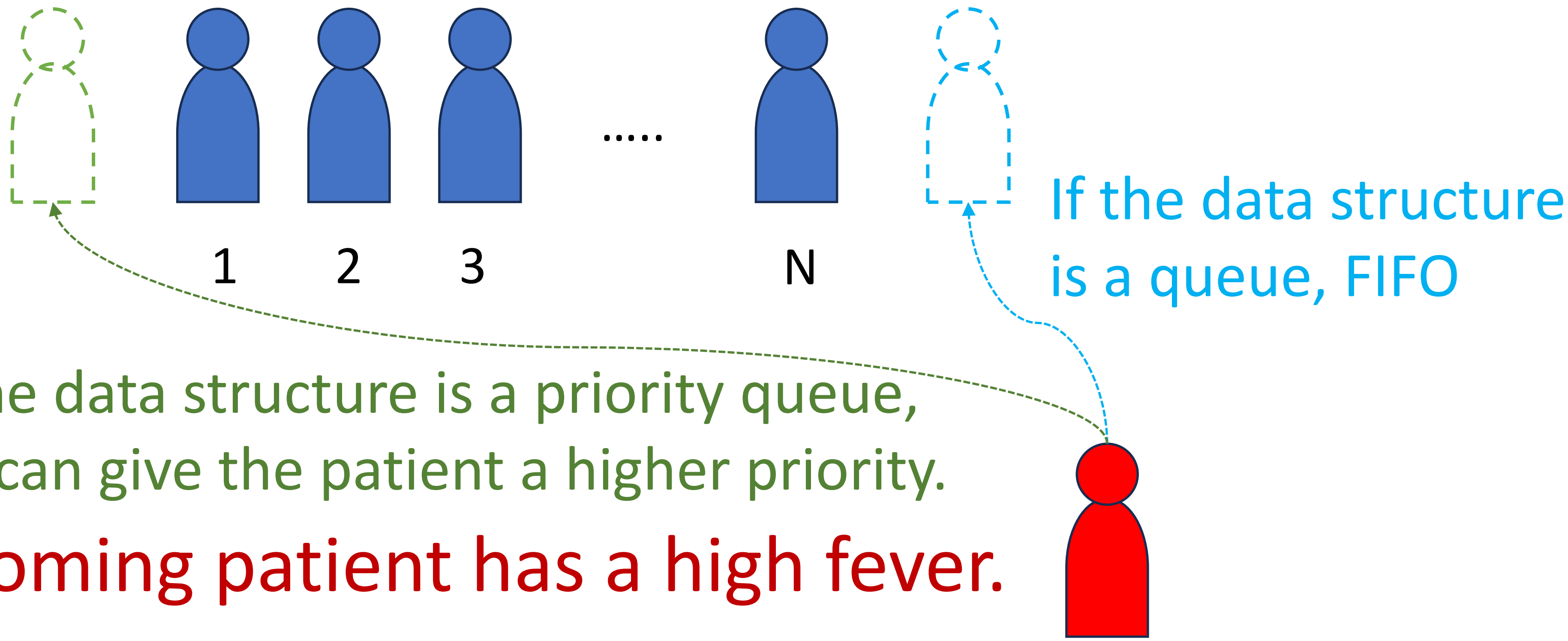
Leftist trees

Ch. 9.2

Priority queues

- In the clinics, patients are waiting.



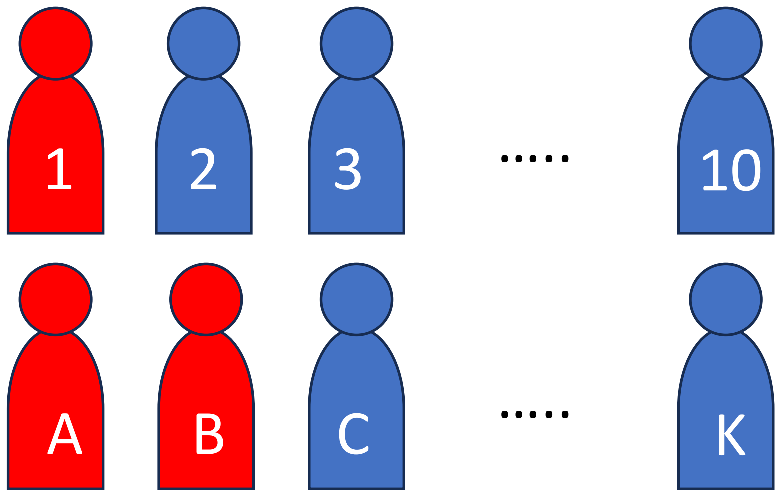
- A new coming patient has a high fever.

Merge two priority queues

- In the clinics, two doctors are on call.

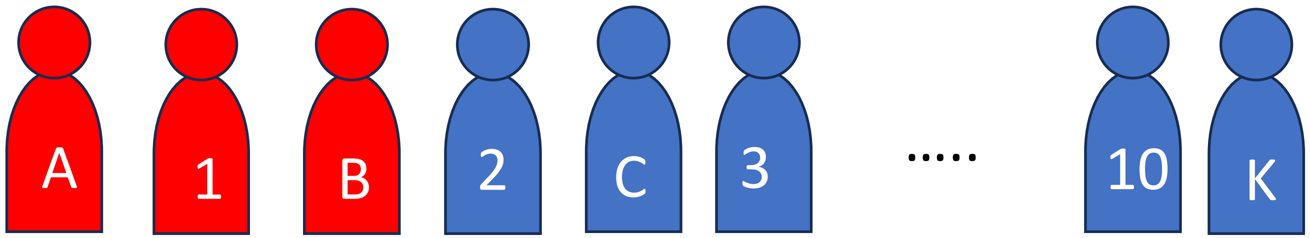
Waiting for Doctor1

Waiting for Doctor2



- When Doctor1 becomes off duty, the priority queues have to be melded together.

Waiting for Doctor2



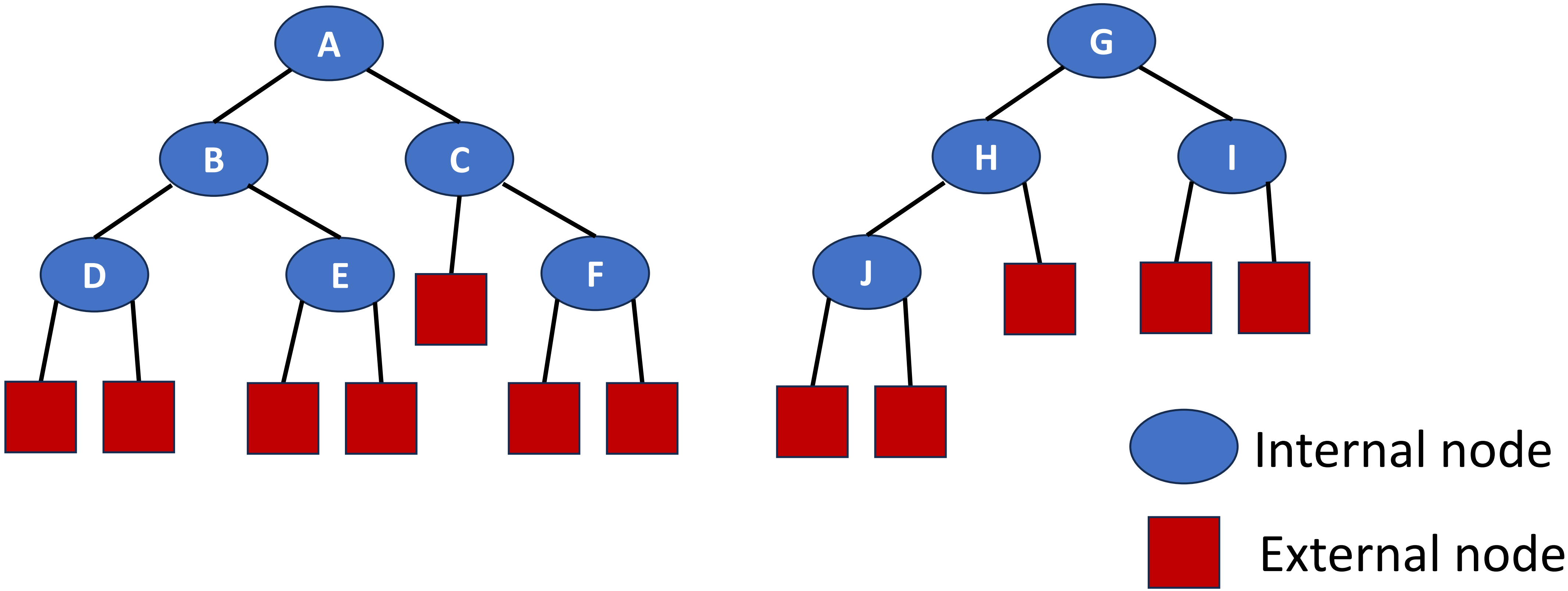
Operation: Meld

- Priority queues implemented using heaps:
  - Inserting the elements of one heap into another heap.
  - By inserting repeatedly, the meld takes O(n log n).
  - By initializing a max heap, the meld takes O(n). (See Ch. 7.6)

- Priority queues implemented using leftist tree:
  - Insertion, deletion: O(log n)
  - Find min or max: O(1)
  - Meld: O(log n)

Extended binary trees

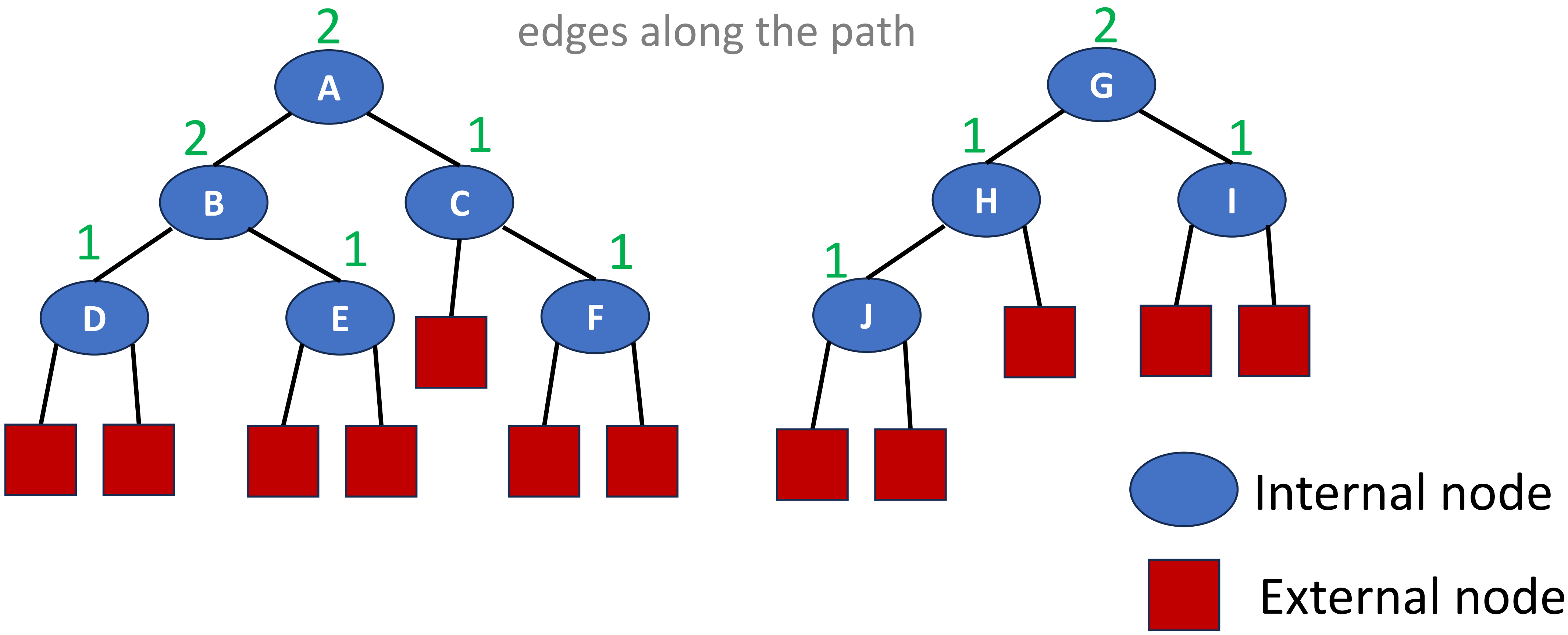
- Start with any binary tree.
- Add an external node wherever there is an empty subtree.
- Obtain an extended binary tree.



Function: shortest(*x*)

- Let *x* be a node in an extended binary tree.
- Function *shortest(x)* is the length of a shortest path from *x* to an external node.

*Path length: number of edges along the path

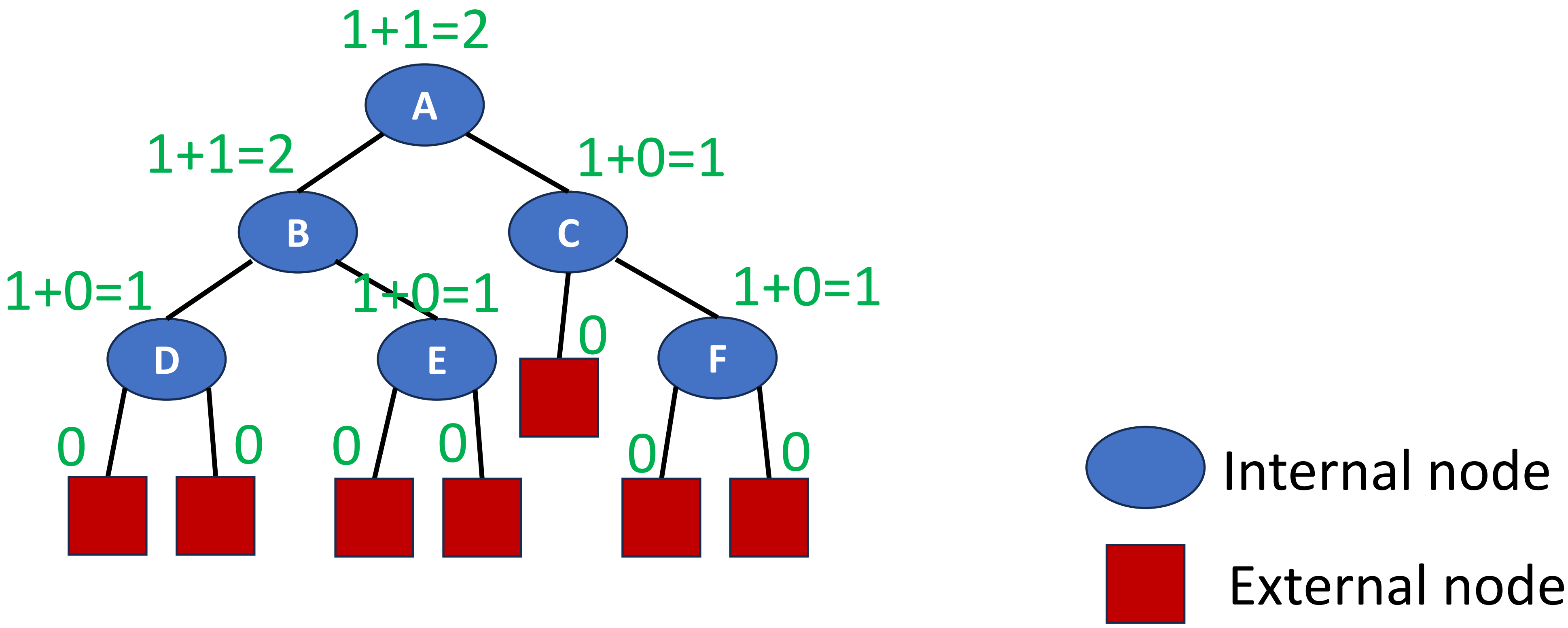


Calculation of shortest(x)

If x is an <u>external node</u>, *shortest*(*x*)=*0* .

Otherwise,

$$shortest(x) = 1 + \min\{shortest(leftChild(x)),\ shortest(rightChild(x))\}\ .$$

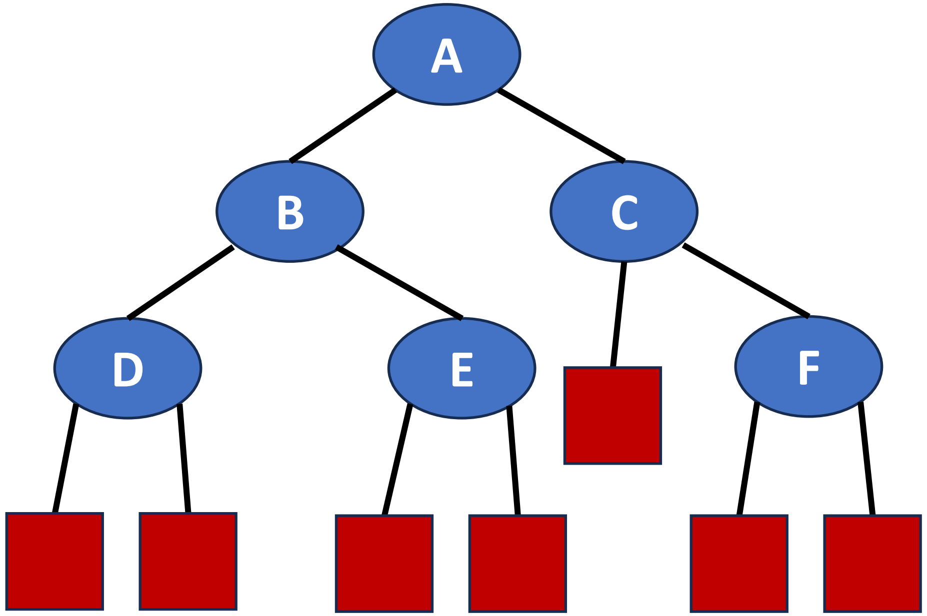


Height biased leftist trees (HBLT)

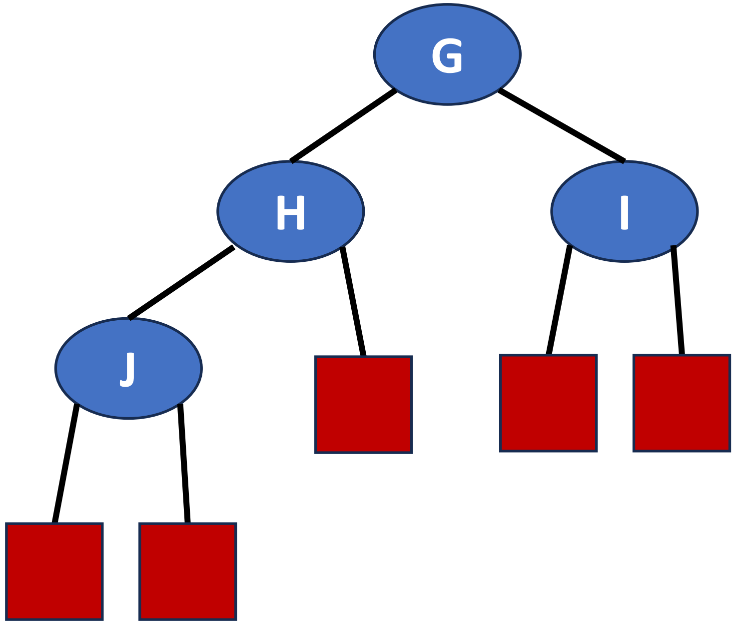
- An extended binary tree
- For every internal node x,

$$shortest(\ leftChild(x)\ ) \geq shortest(\ rightChild(x)\ )$$

Is this a leftist tree?

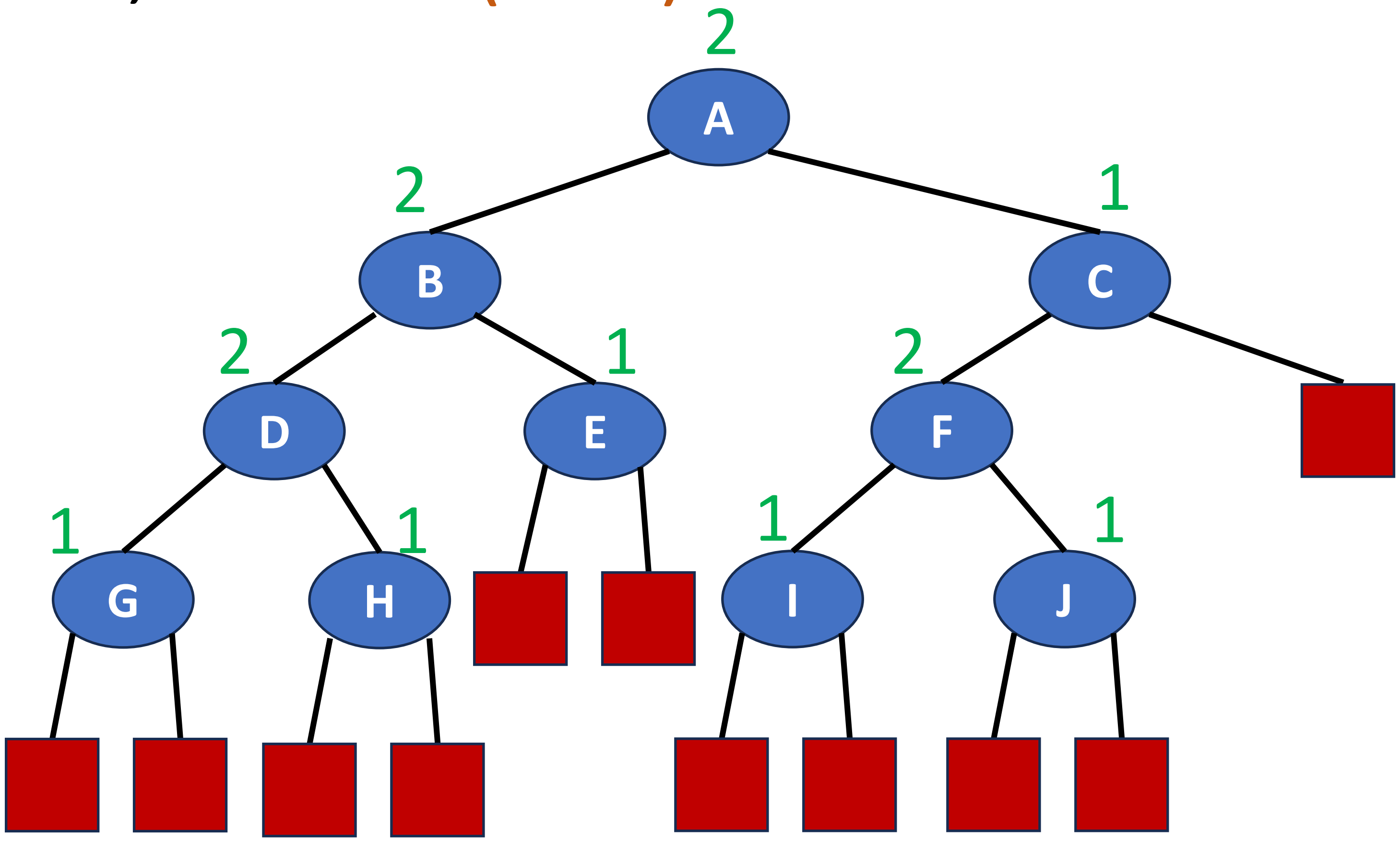


Is this a leftist tree?



Properties of leftist trees (1)

- The rightmost path is a shortest root to external node path, that is, *shortest*(*root*).

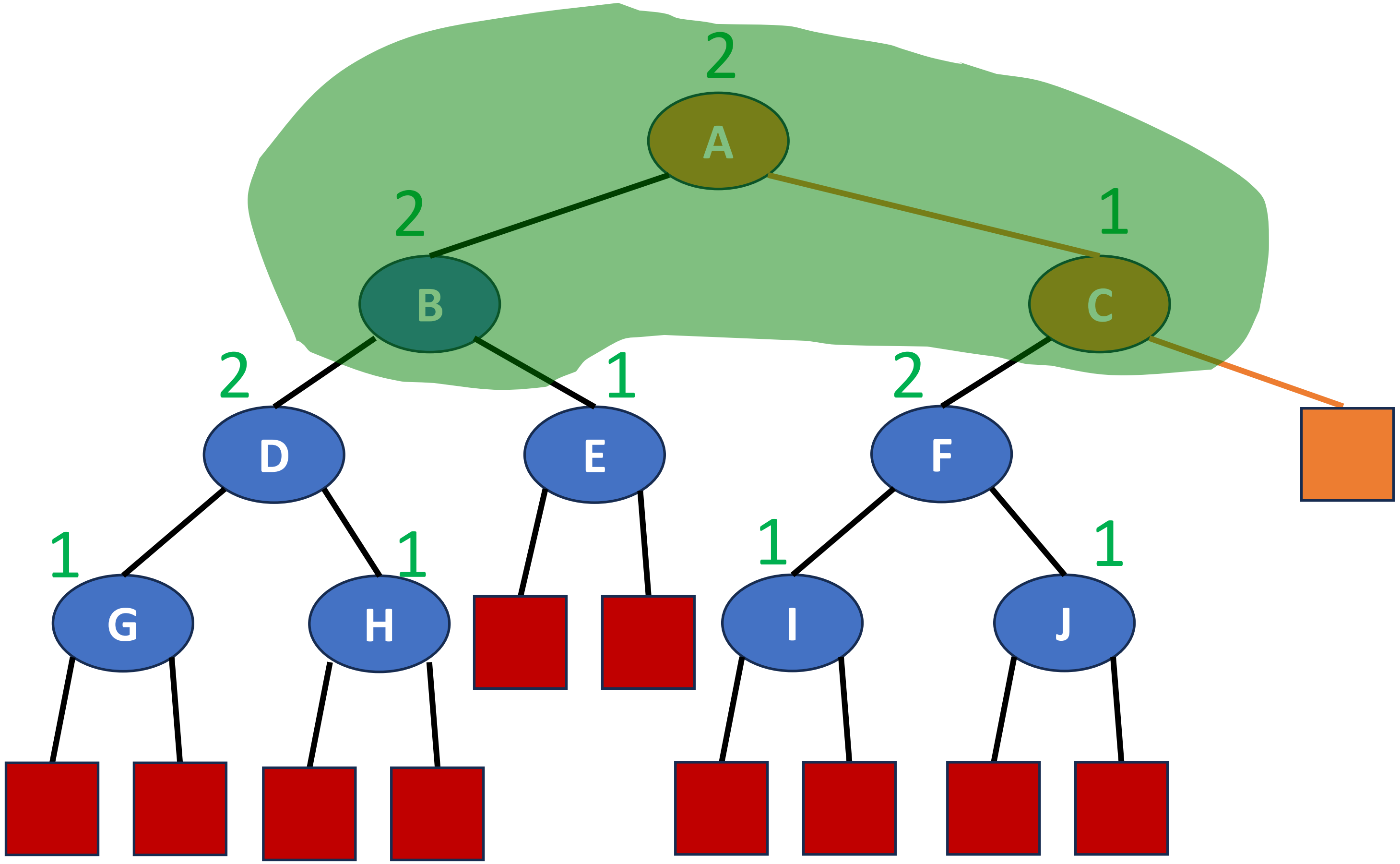


Properties of leftist trees (2)

- Number of internal nodes $n \geq 2^{shortest(root)}-1$

$$shortest(root) \leq \log_2(n+1)$$

The first *shortest*(*root*) levels of nodes constitute a full binary tree.



Level	Number of nodes
1	2^0
2	2^1

Total: $2^0 + 2^1$

For a *shortest*(*root*)-level full binary tree, the total number of nodes is

$$\sum_{i=1}^{shortest(root)} 2^{i-1} = 2^{shortest(x)} - 1$$

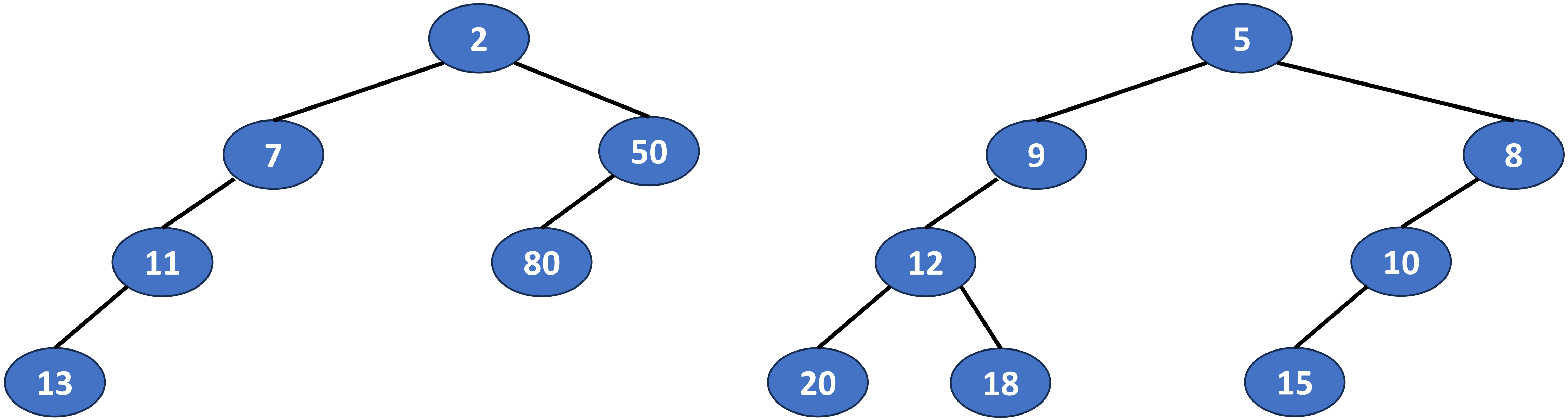
Properties of leftist trees (3)

- Length of rightmost path is $O(\log n)$, where n is the number of nodes in a leftist tree.

Leftist trees as priority queues

- Min leftist tree: Leftist tree that is a min tree.
  - Used as a min priority queue.
- Max leftist tree: Leftist tree that is a max tree.
  - Used as a max priority queue.

Examples of min leftist trees



Operations for min leftist trees

- Insert(x, T_a):
  - Create a min leftist tree T_b containing only x
  - Meld T_a and T_b

- DeleteMin(T_a):
  - Meld the two min leftist trees *root->leftChild* and *root->rightChild*
  - Delete the original *root*

- Meld(T_a, T_b)

- Initialize()

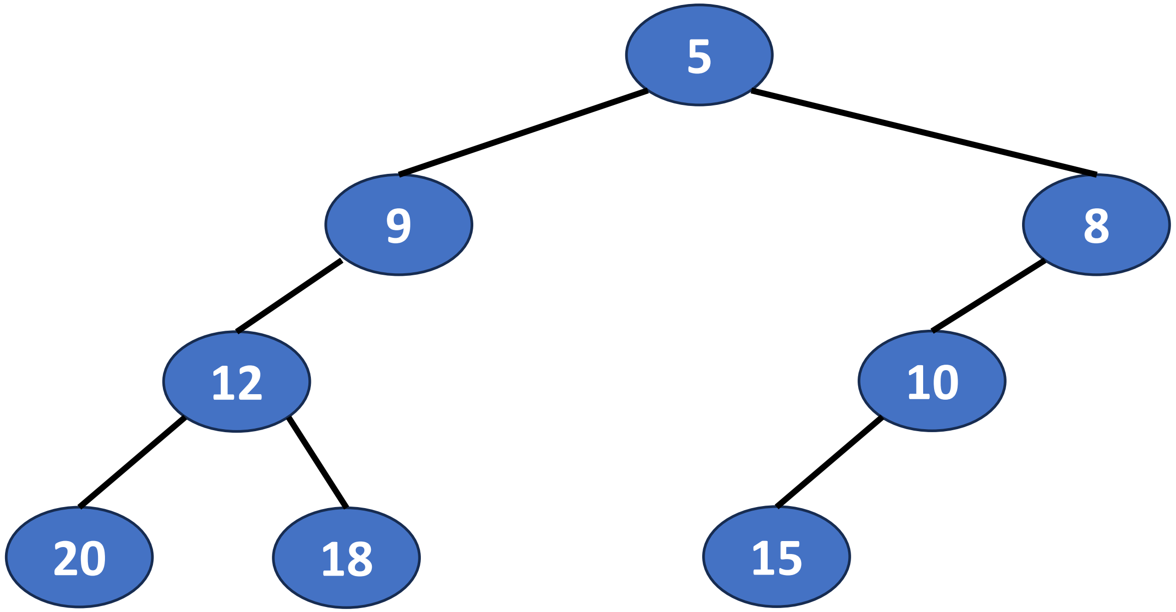
Operation: Insert()

- Example: insert(7, T_a)

  STEP 1: Create a single node min leftist tree T_b .

  STEP 2: Meld the two min leftist trees.

T_b



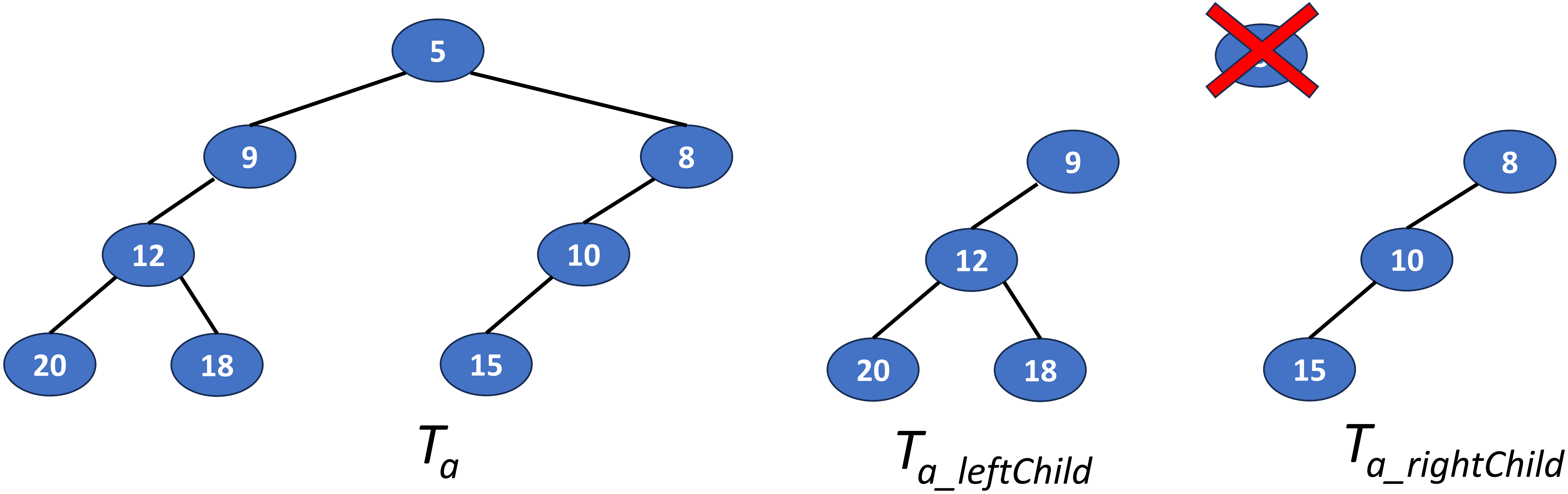
T_a

Operation: DeleteMin()

- Example: deleteMin(T_a)

  STEP 1: Get subtrees of root, $T_{a\_leftChild}$ and $T_{a\_rightChild}$

  STEP 2: Delete the root and meld the two subtrees.



Operation: Meld(T_a, T_b)

Maintaining property of min tree:
Root of a subtree has the smallest key.

- **Phase 1: Top-down process**

  Going down along the rightmost paths in T_a or T_b
  - Comparing their roots.
    For the tree with smaller root, going down to its right child.
    - If no right child, attaching another tree as right subtree.
    - If having right child, comparing again.

Maintaining property of leftist tree:
Shortest(leftchild)>shortest(rightchild).

- **Phase 2: Bottom-up process**

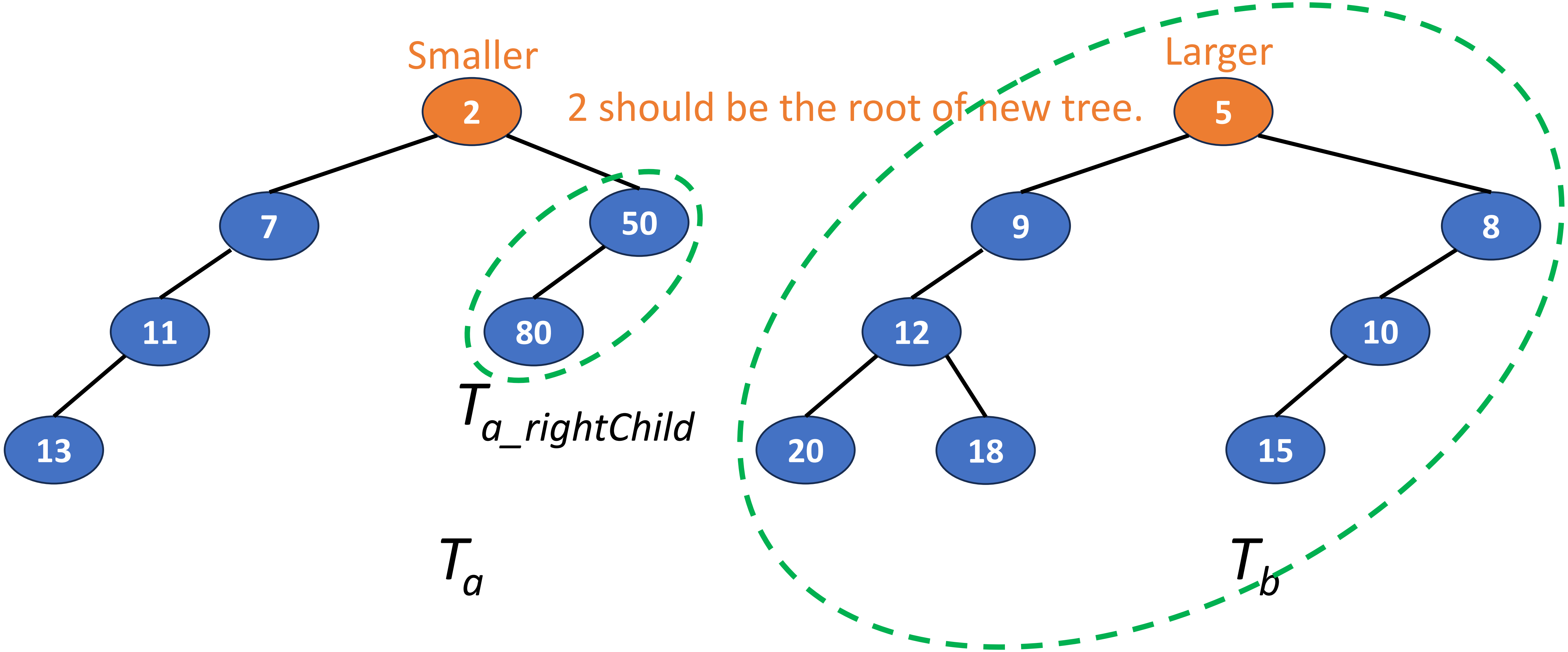
  Climbing up through the rightmost path of the new tree
  - If *shortest*(*node->leftChild*) < *shortest*(node->*rightChild*), interchanging the left and right subtrees of the node.

Example of meld operation

- Phase 1: Top-down

  STEP 1: Comparing 2 and 5.

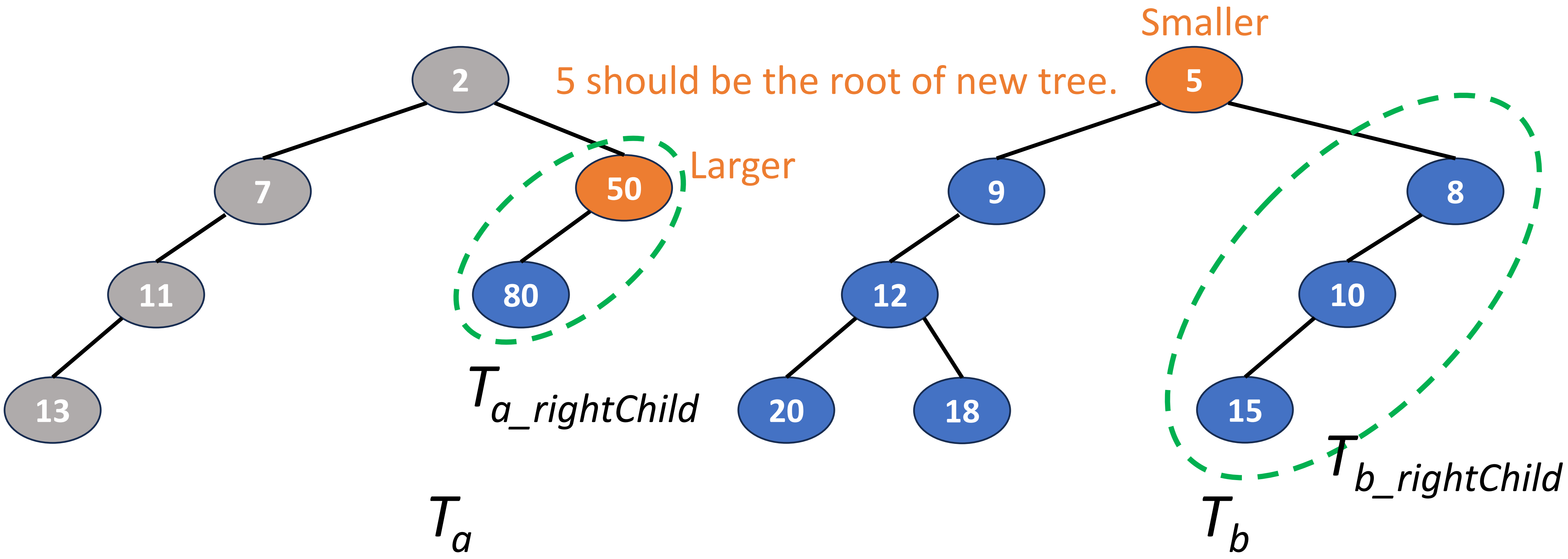
Meld subtree $T_{a\_rightChild}$ and T_b



Example of meld operation

- Phase 1: Top-down

STEP 2: Comparing 50 and 5. *Meld subtree $T_{a\_rightChild}$ and $T_{b\_rightChild}$*

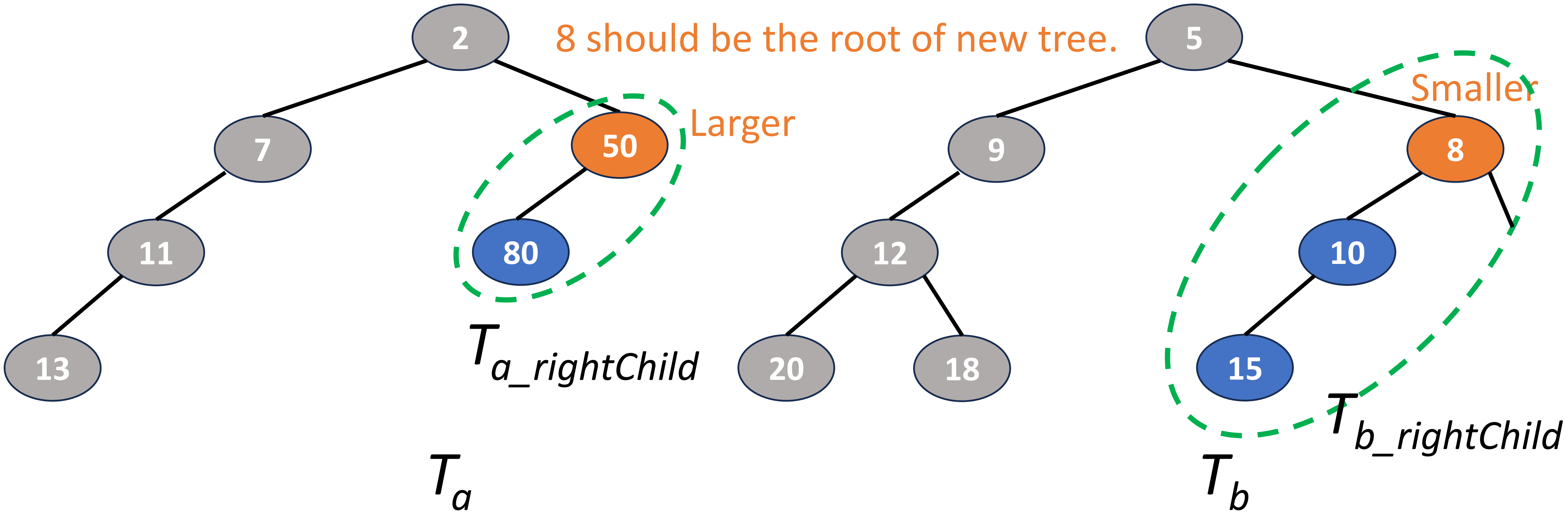


Example of meld operation

- Phase 1: Top-down

  STEP 3: Comparing 8 and 50.

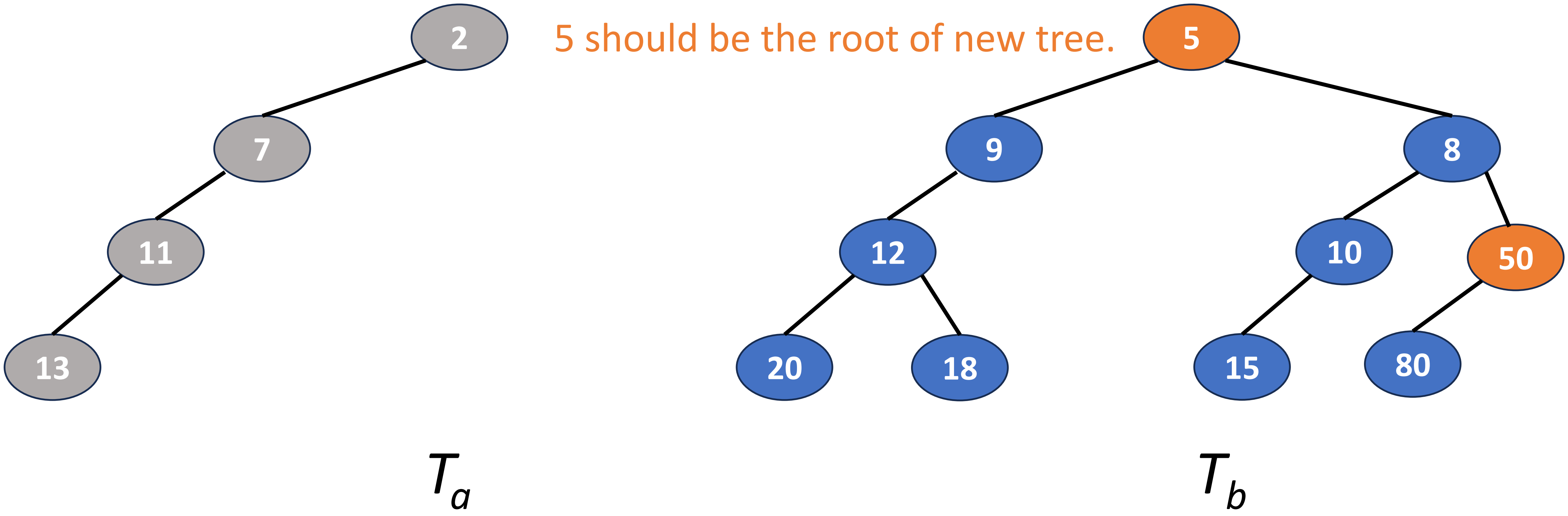
Attach $T_{a\_rightchild}$ as right subtree of $T_{b\_rightChild}$



Example of meld operation

- Phase 1: Top-down

  STEP 2: Comparing 50 and 5. *Meld subtree $T_{a\_rightChild}$ and $T_{b\_rightChild}$*



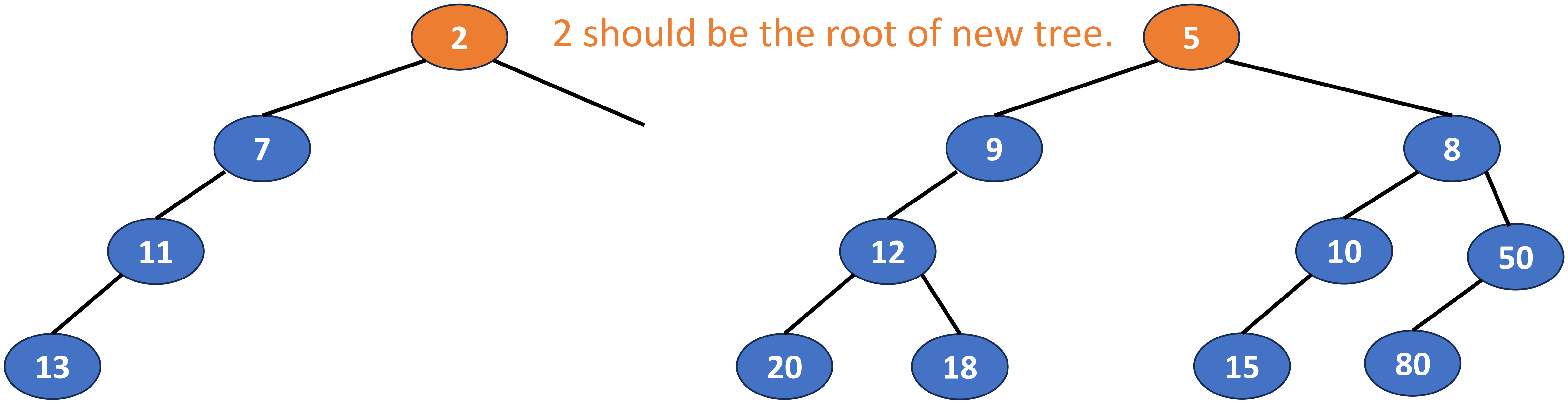
Example of meld operation

- Phase 1: Top-down

  STEP 1: Comparing 2 and 5.

Meld subtree $T_{a\_rightChild}$ and T_b

2 should be the root of new tree.

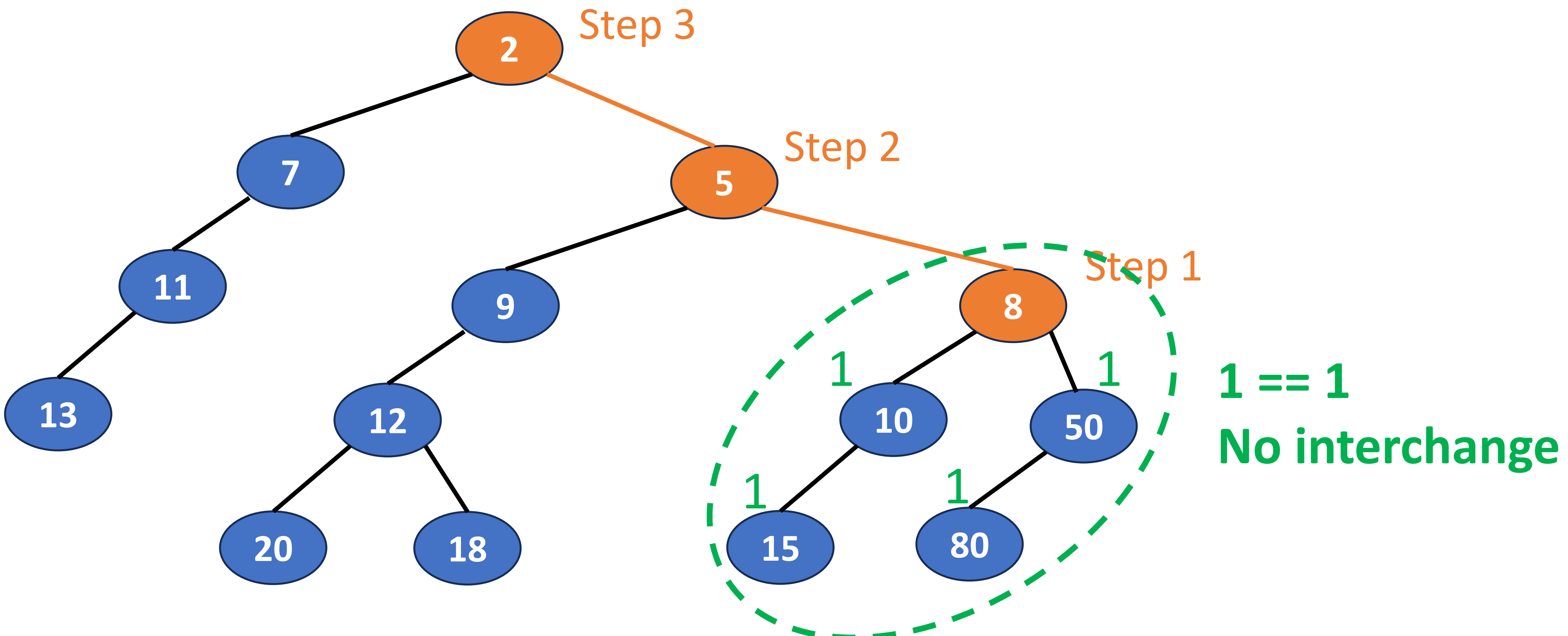


T_a T_b

Example of meld operation

- Phase 2: Bottom up

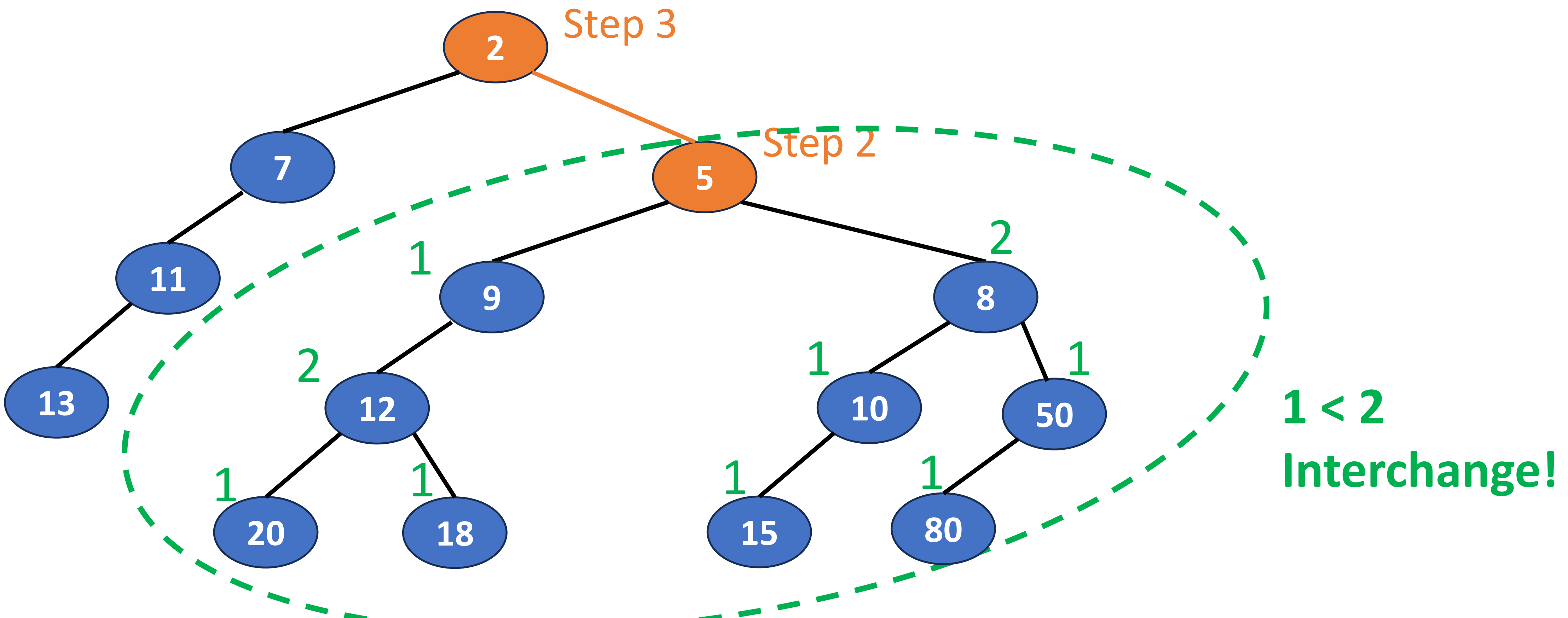
  STEP 1: Starting from last modified root (node containing 8).
  Comparing *shortest*(*leftChild*) and *shortest*(*rightChild*).



Example of meld operation

- Phase 2: Bottom up

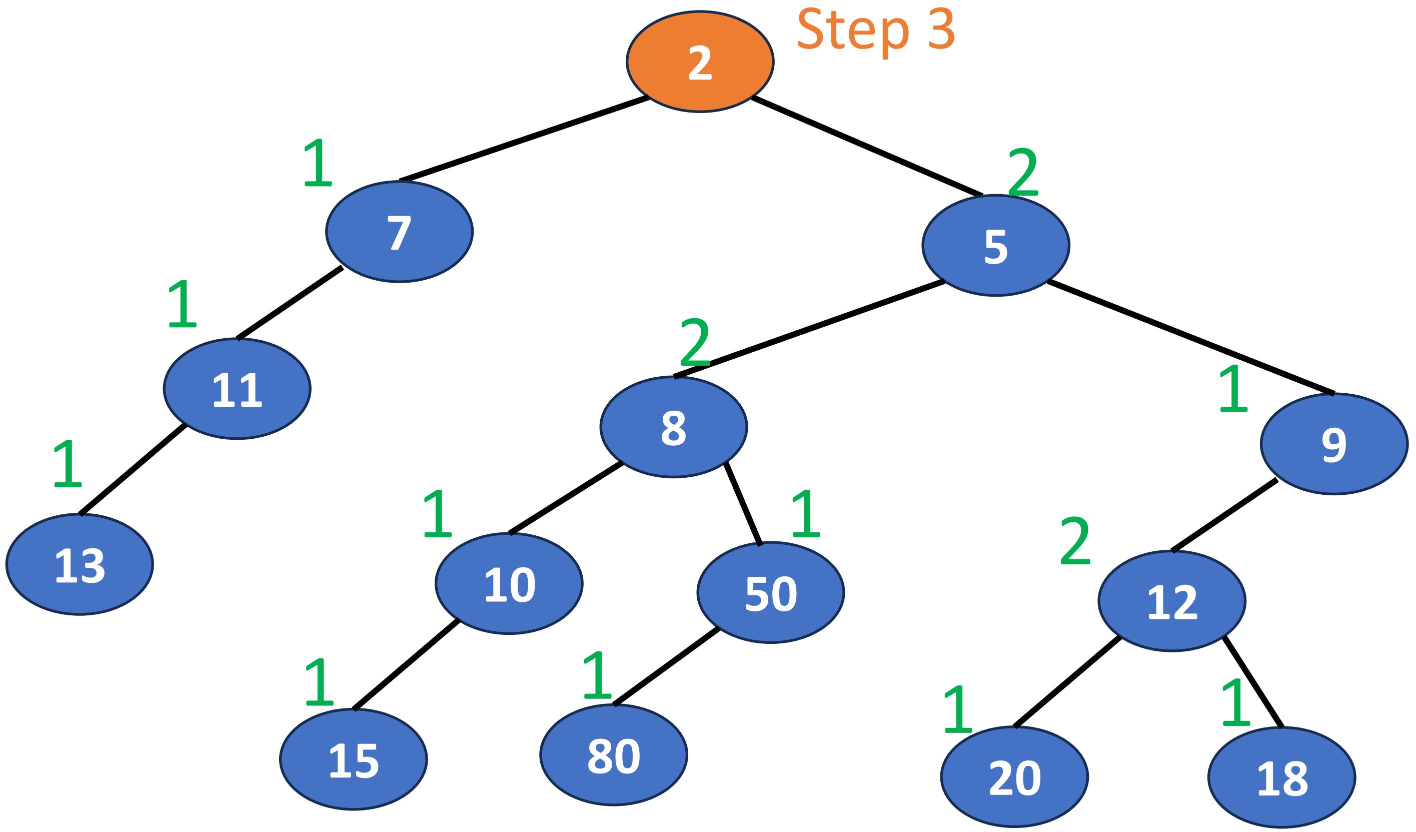
  STEP 2: Comparing *shortest*(*leftChild*) and *shortest*(*rightChild*) of the node containing 5.



Example of meld operation

- Phase 2: Bottom up

  STEP 3: Comparing *shortest*(*leftChild*) and *shortest*(*rightChild*) of the root.



1 < 2

Interchange!

Complexity for meld operation

- Length of rightmost path is O(log n), where *n* is the number of nodes in a leftist tree.

- A meld operation moves down and climbs up along the rightmost paths of the two leftist trees.

- Time complexity for meld: log(*m*)

  m: number of total elements in two leftist trees.

Operation: Initialization()

- Create *n* single node min leftist trees and place them in a FIFO queue
- Repeatedly remove two min leftist trees from the FIFO queue, meld them, and put the resulting min leftist tree into the FIFO queue
- The process terminates when only 1 min leftist tree remains in the FIFO queue

Complexity: $\underbrace{n}_{\text{Create } n \text{ single node trees}} + \underbrace{2}_{\text{Top-down + bottom-up}} \times \left(\underbrace{1 \times \frac{n}{2}}_{\text{Meld two h=1 trees for n/2 times}} + \underbrace{2 \times \frac{n}{4}}_{\text{Meld two h=2 trees for n/4 times}} + \underbrace{3 \times \frac{n}{8}}_{\text{Meld two h=3 trees for n/8 times}} + \cdots \right) = O(n)$

Weighted-based leftist trees (WBLT)

Please reply your answers of Q3 via the following link:

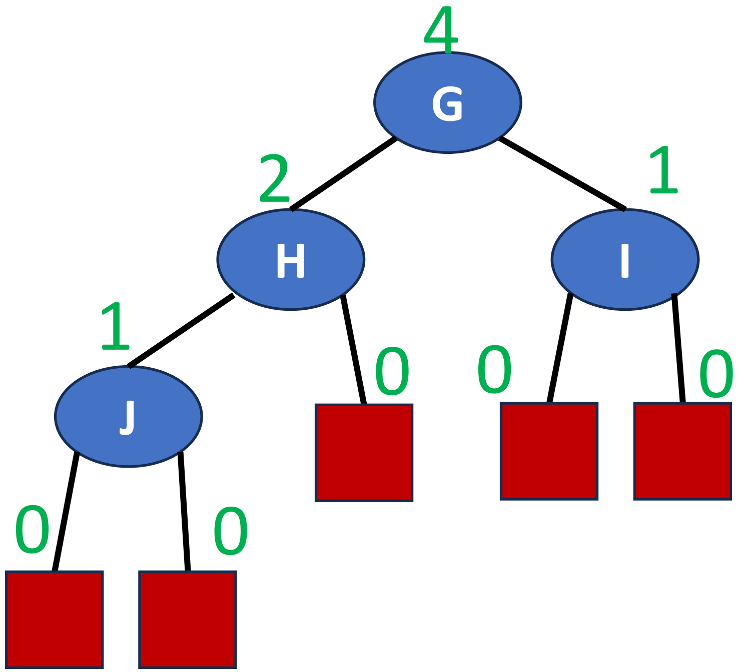
Group members: 1~3 people

- An extended binary tree
- For every internal node x,

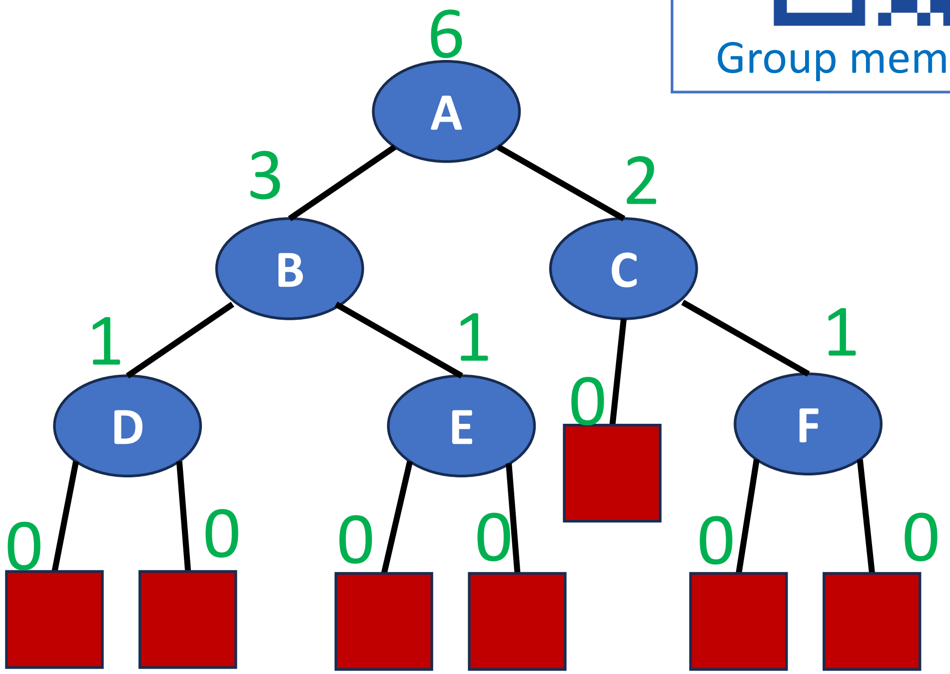
Count(): number of internal nodes

$$count(\ leftChild(x)\ ) \geq count(\ rightChild(x)\ )$$

Is this a WBLT?



Q3: Is this a WBLT?



Summary

- Meldable priority queue
  - Implemented using heaps: O(n)
  - Implemented using leftist trees: O(log n)

- Operations of leftist trees
  - Insert(), deleteMin(), meld(), initialization()