Binary Search Trees

Kuan-Yu Chen (陳冠宇)

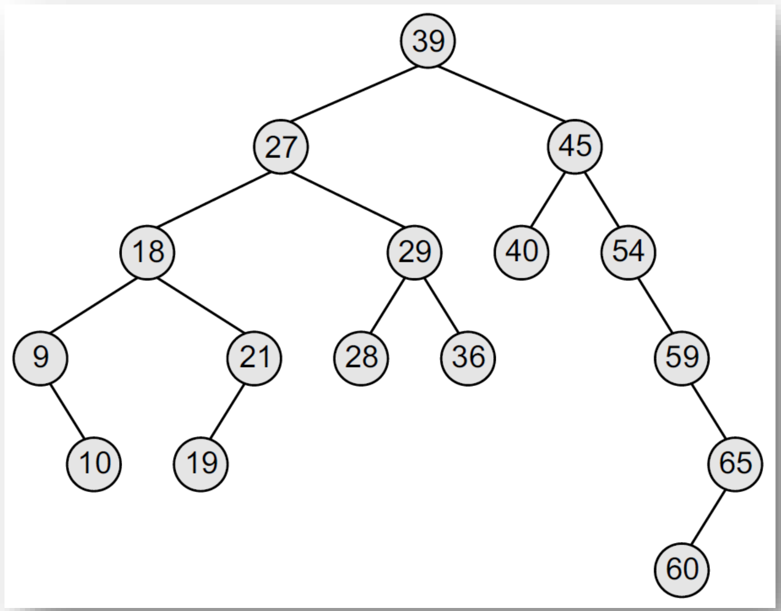
2018/10/15 @ TR-212, NTUST

Review

- Trees
  - General Trees
  - Forests
  - Binary Trees
  - Binary Search Trees
  - Expression Trees
  - Tournament Trees

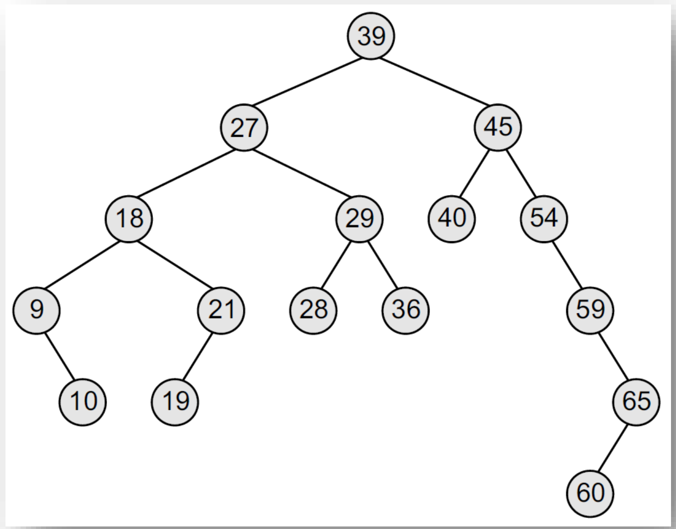
Binary Search Trees.

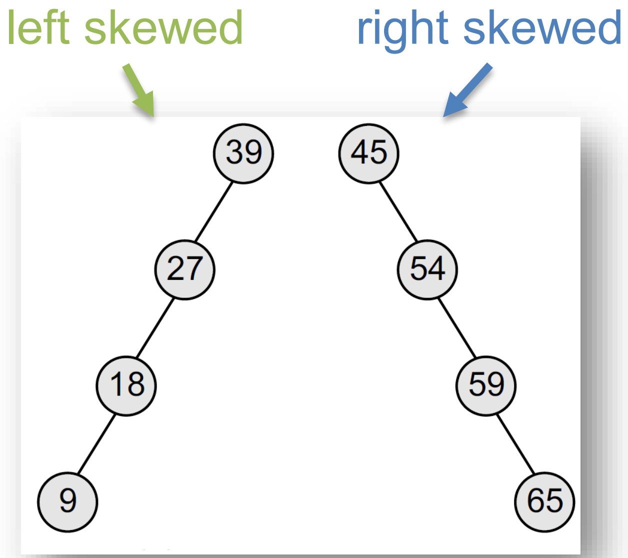
- A binary search tree, also known as an **ordered binary tree**, is a variant of binary trees in which the nodes are arranged in an order
  - All the nodes in the **left sub-tree** have a value **less** than that of the root node
  - All the nodes in the **right sub-tree** have a value either **equal to or greater** than the root node



Binary Search Trees..

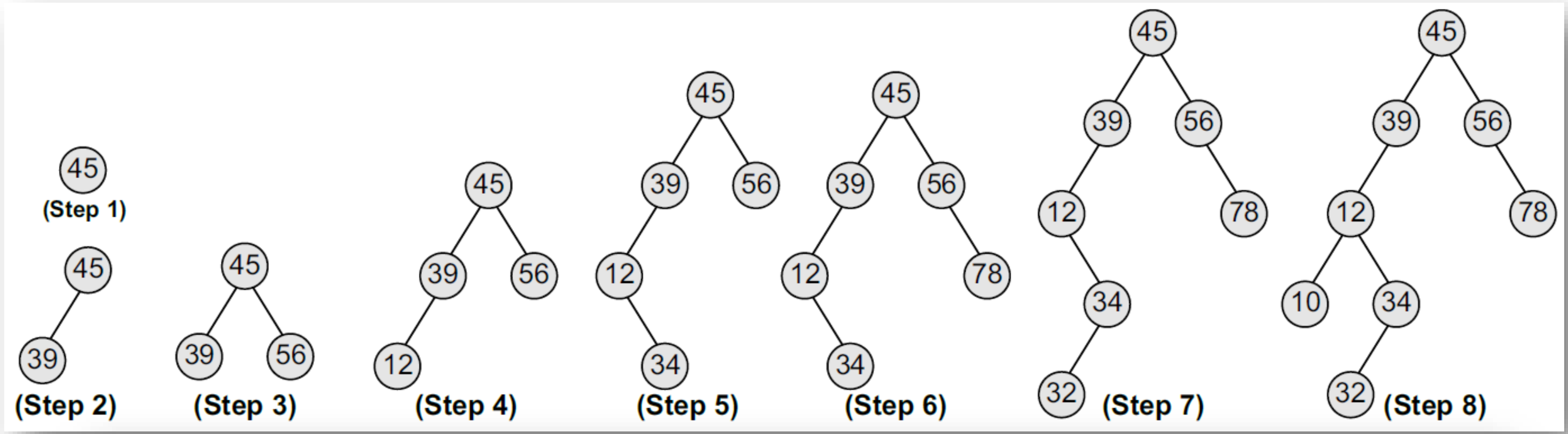
- Since the nodes in a binary search tree are ordered, the time needed to search an element in the tree is greatly reduced
  - We do not need to traverse the entire tree
  - At every node, we get a hint regarding which sub-tree to search in
    - The average running time of a search operation is $O(log_2 n)$
    - In the worst case, a binary search tree will take $O(n)$ time to search for an element

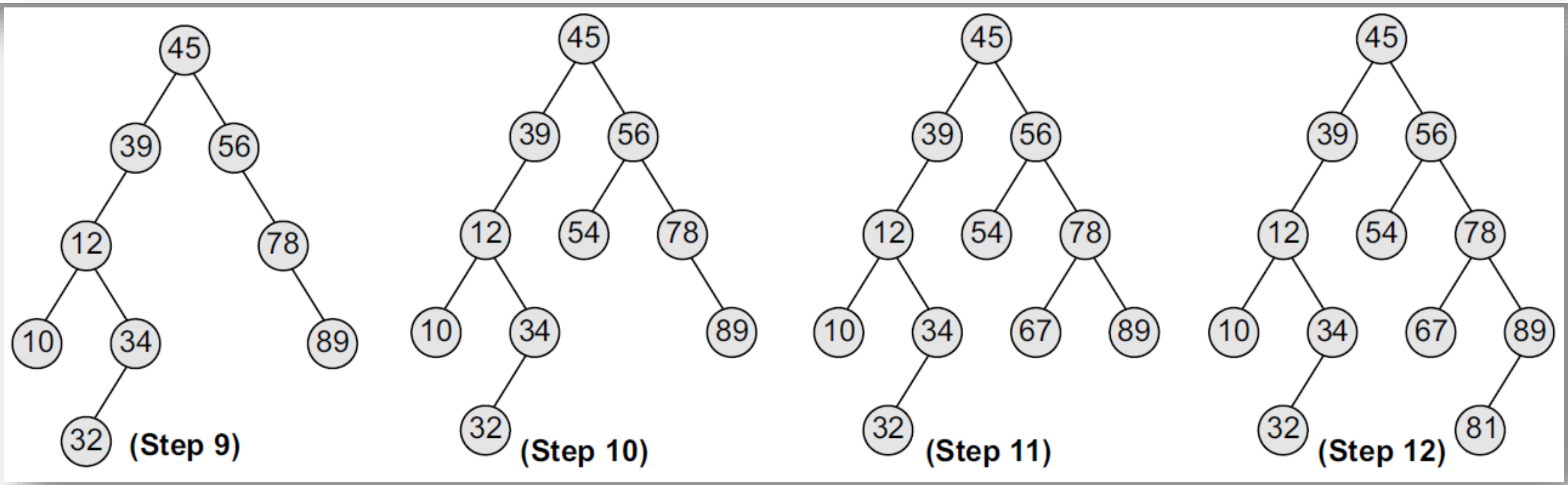




Steps for Creating a Binary Search Tree

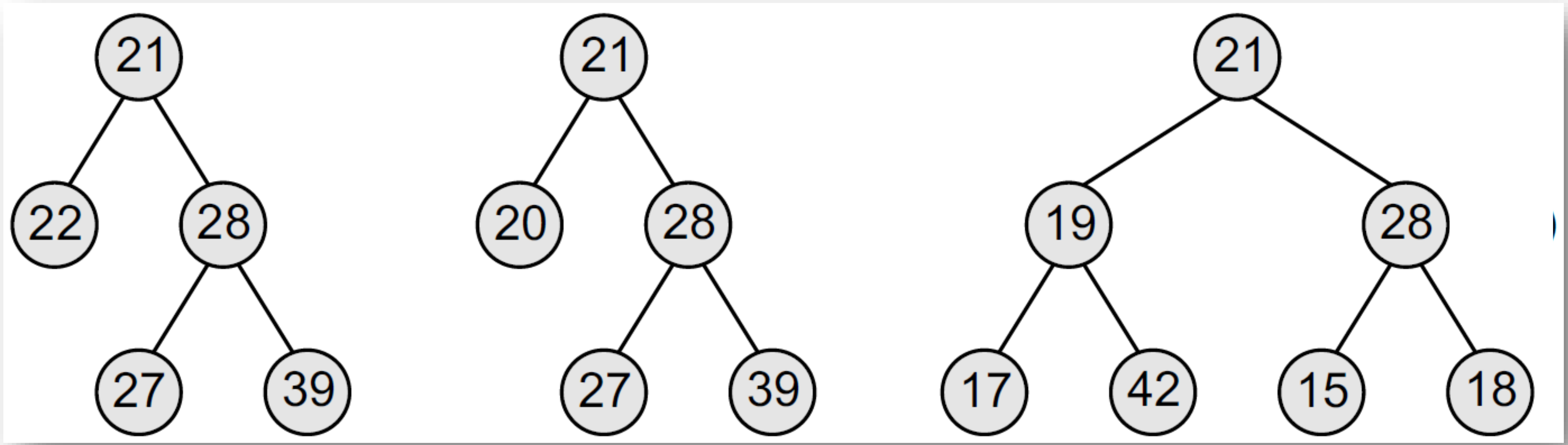
- Create a binary search tree using the following data elements:
45, 39, 56, 12, 34, 78, 32, 10, 89, 54, 67, 81





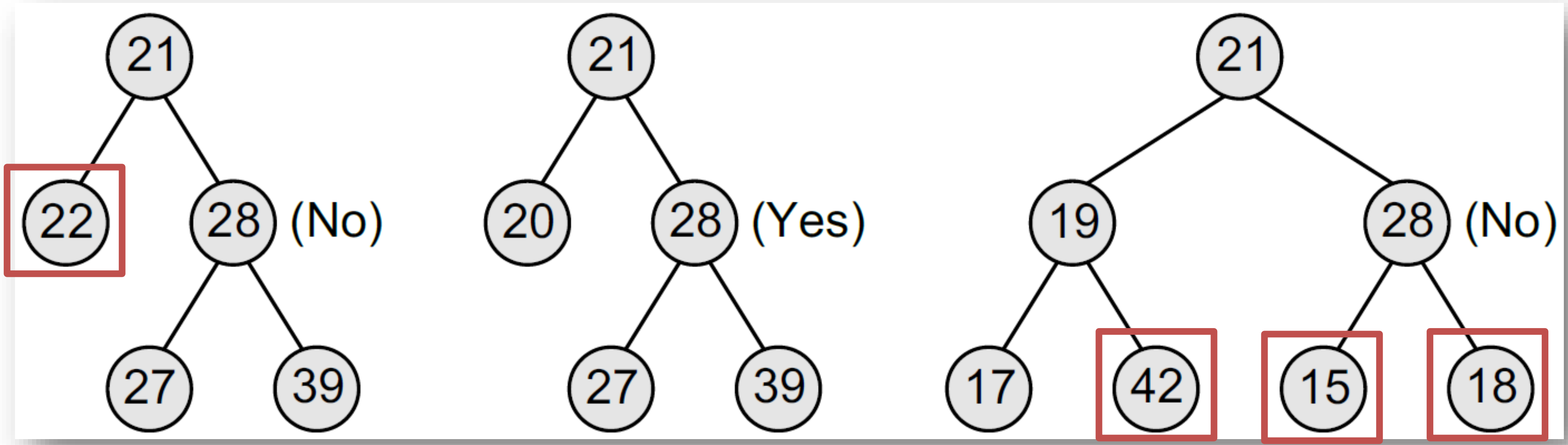
Binary Search Trees or not?.

- Which trees are binary search trees?



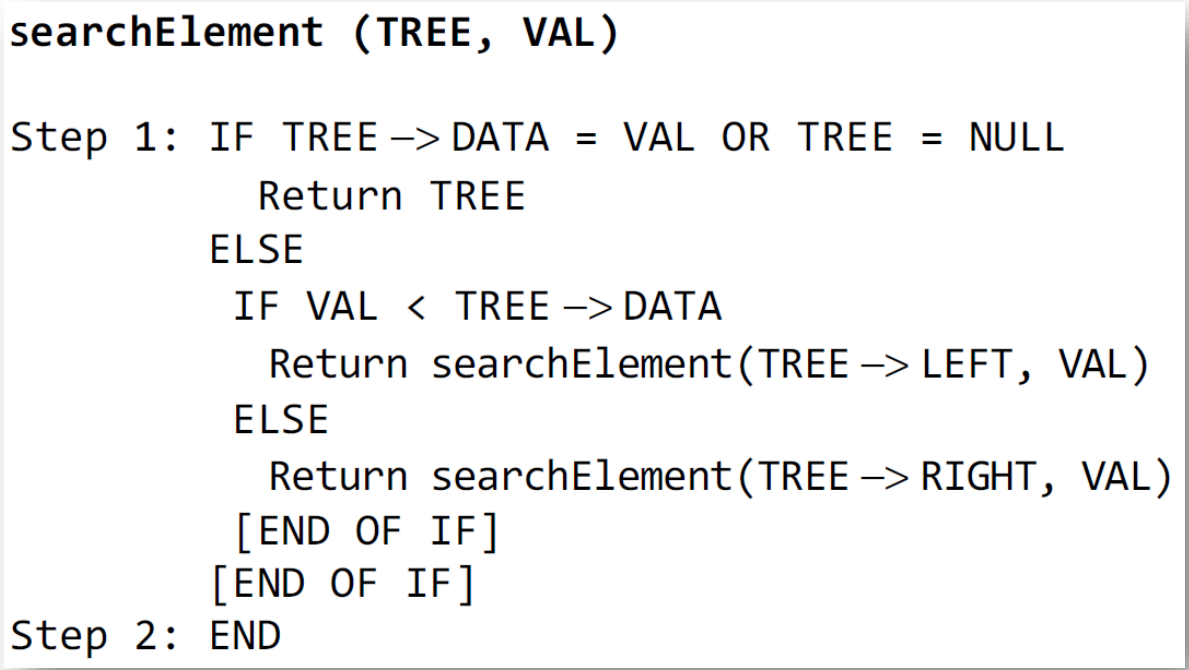
Binary Search Trees or not?..

- Which trees are binary search trees?



Searching a Node.

- The search function is used to find whether a given value is present in the tree or not
  - Checks if the binary search tree is empty
  - Compare the value
    - Find
    - Go left
    - Go right

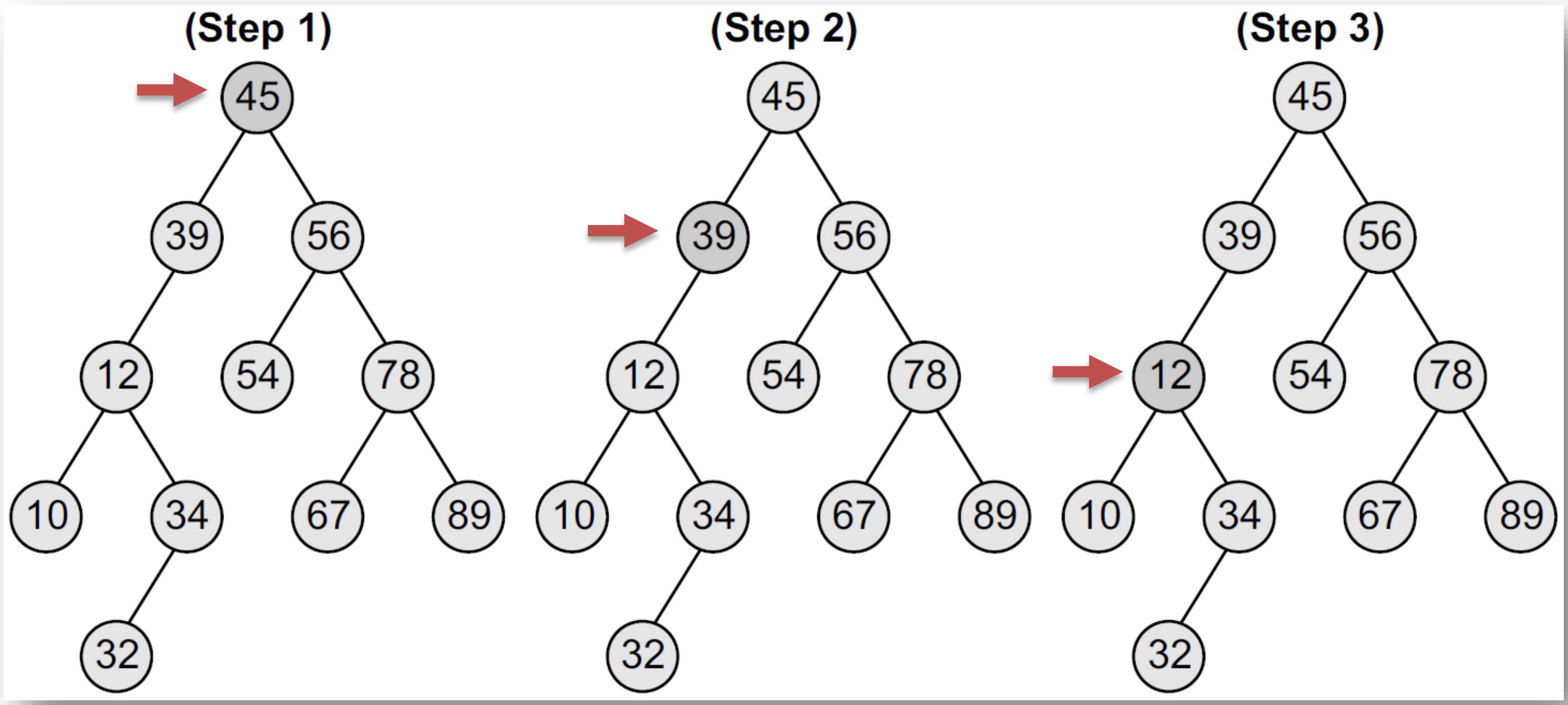


```
searchElement (TREE, VAL)

Step 1: IF TREE –> DATA = VAL OR TREE = NULL
          Return TREE
        ELSE
         IF VAL < TREE –> DATA
           Return searchElement(TREE –> LEFT, VAL)
         ELSE
           Return searchElement(TREE –> RIGHT, VAL)
         [END OF IF]
        [END OF IF]
Step 2: END
```

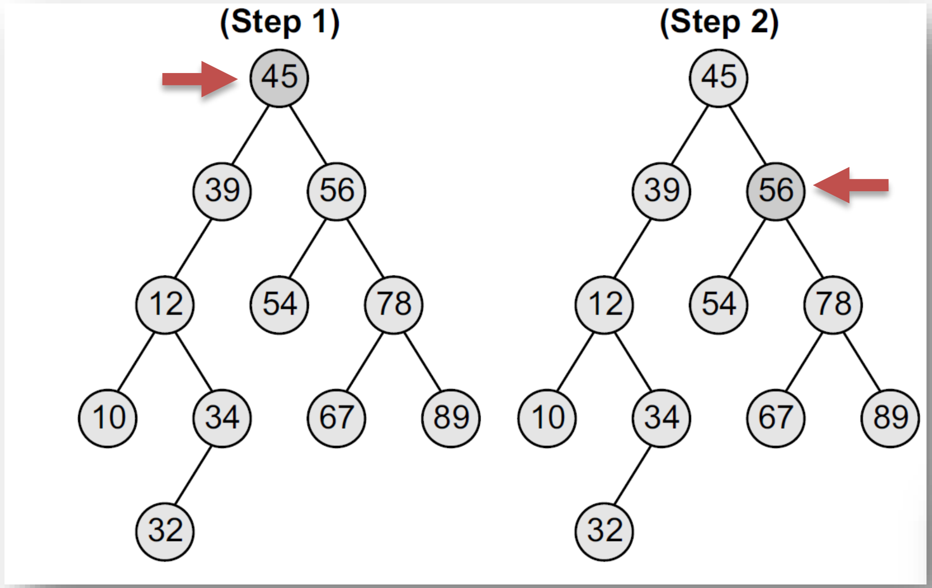
Searching a Node..

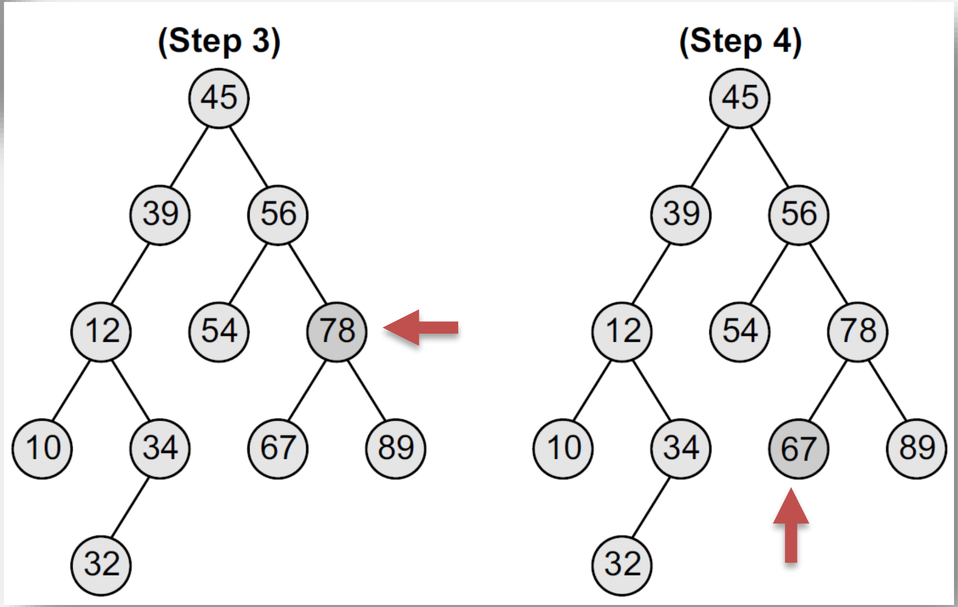
- Searching a node with value 12 in the given binary search tree



Searching a Node...

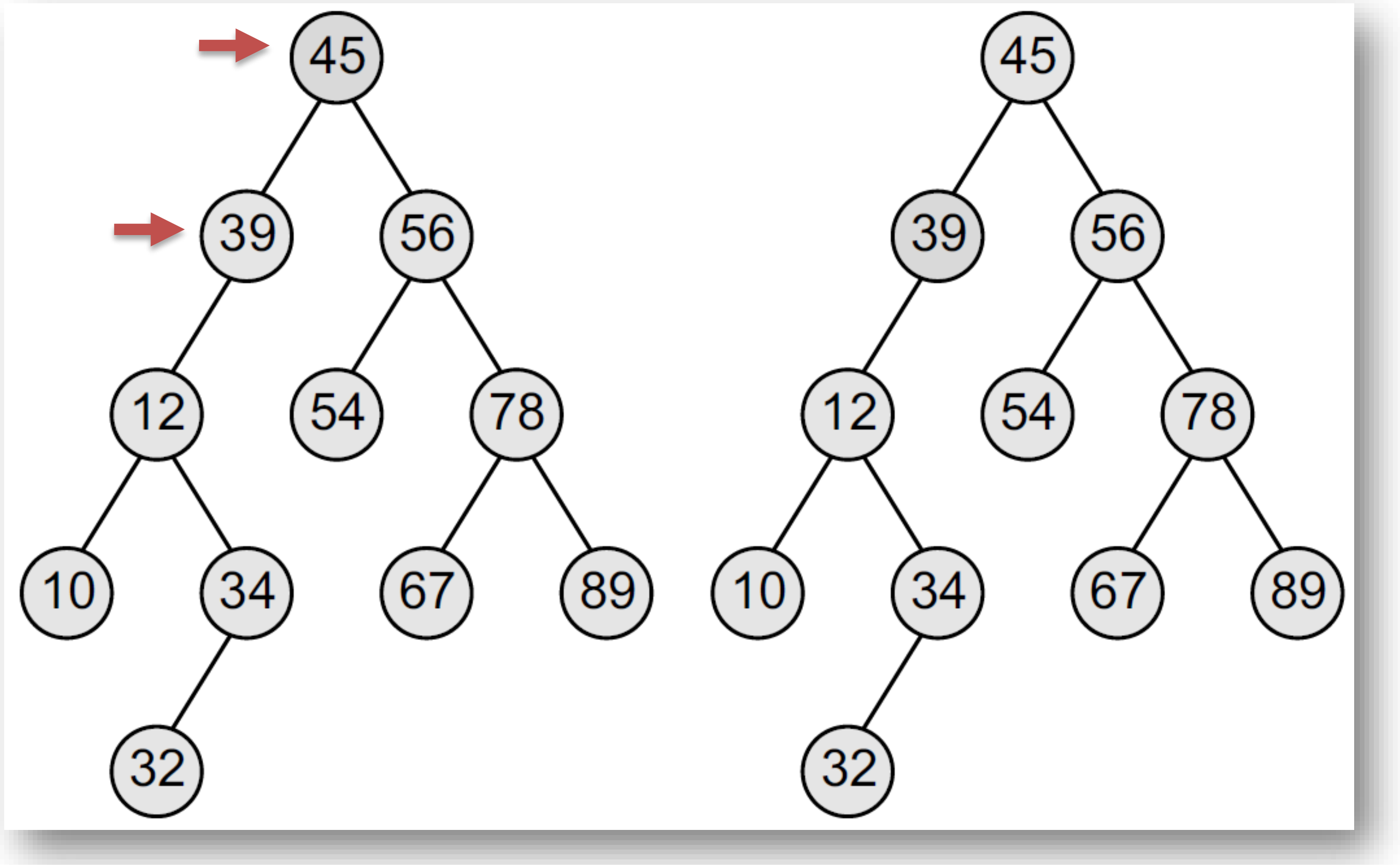
- Searching a node with value 67





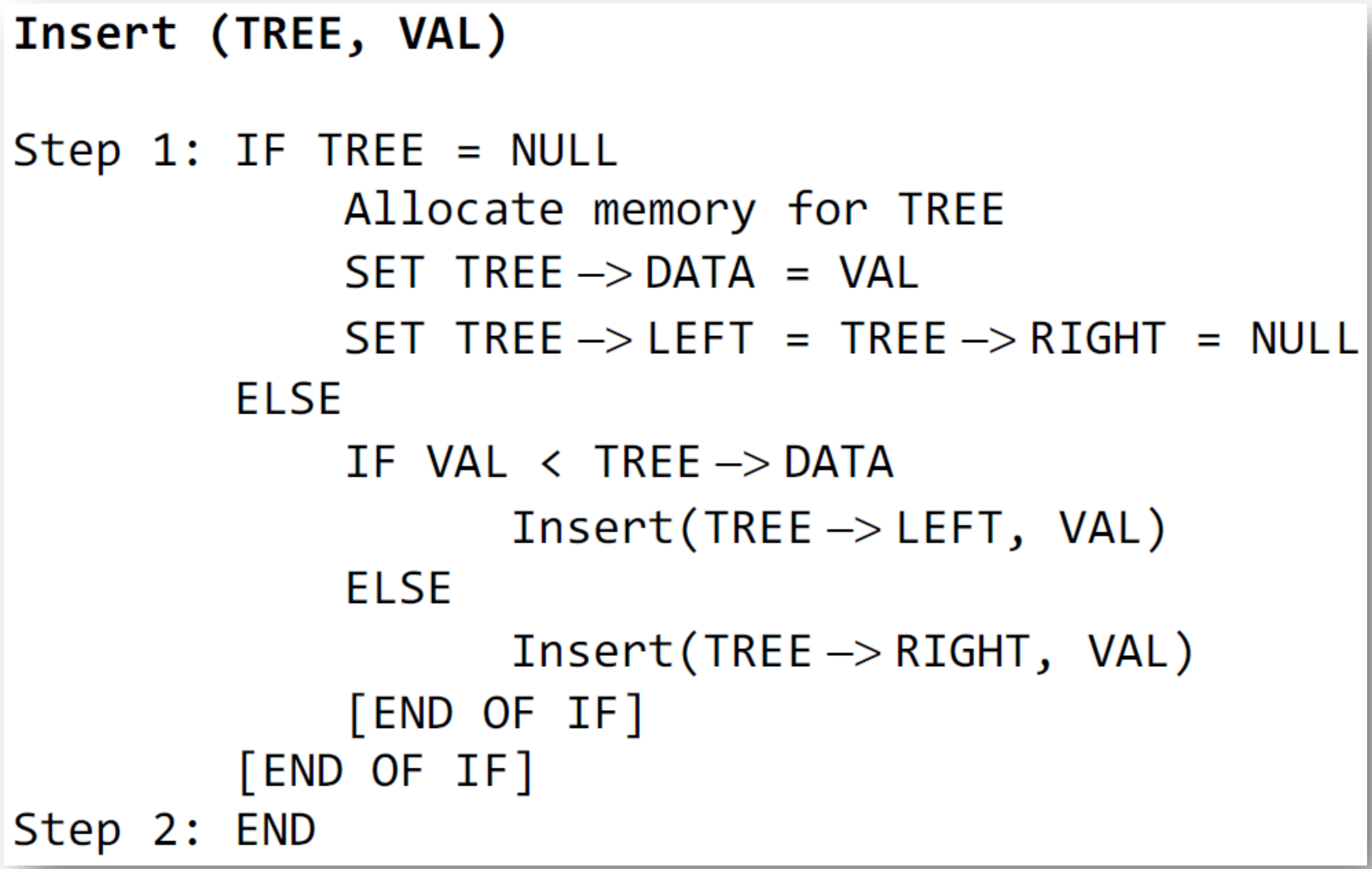
Searching a Node….

- Searching a node with the value 40



Inserting a Node.

- The insert function is used to add a new node with a given value at the correct position in the binary search tree

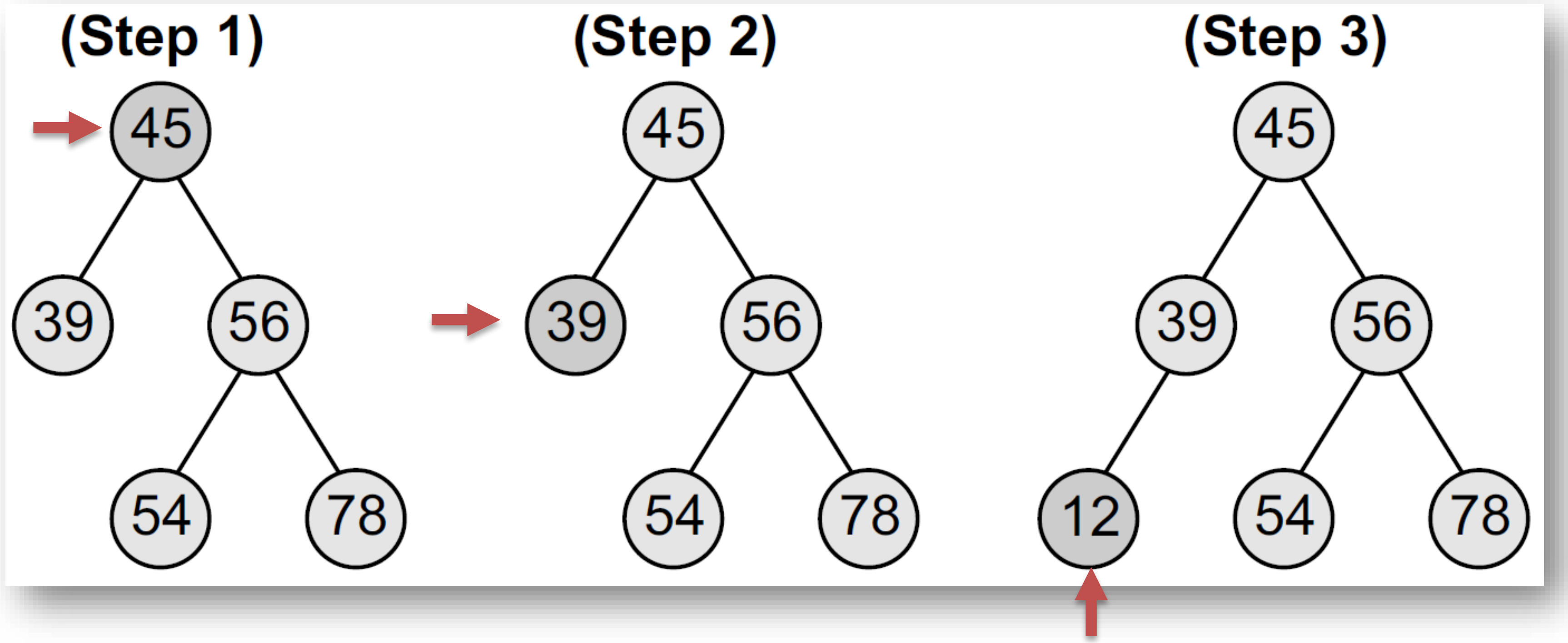


```
Insert (TREE, VAL)

Step 1: IF TREE = NULL
              Allocate memory for TREE
              SET TREE -> DATA = VAL
              SET TREE -> LEFT = TREE -> RIGHT = NULL
        ELSE
              IF VAL < TREE -> DATA
                    Insert(TREE -> LEFT, VAL)
              ELSE
                    Insert(TREE -> RIGHT, VAL)
              [END OF IF]
        [END OF IF]
Step 2: END
```

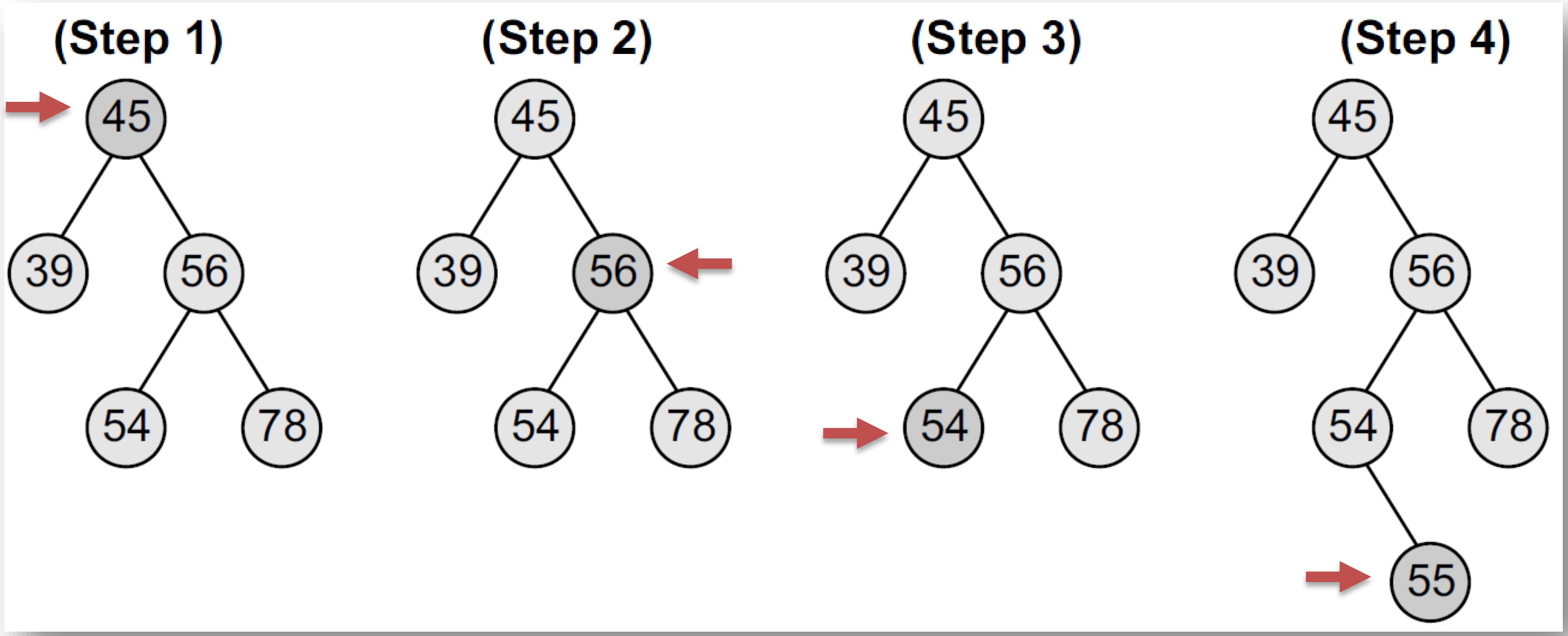
Inserting a Node..

- Inserting a node with values 12



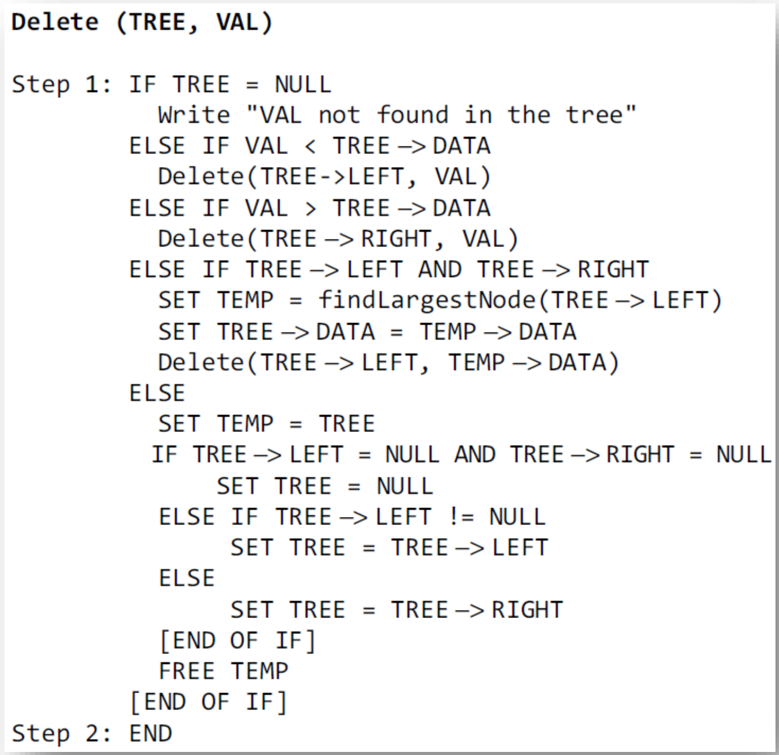
Inserting a Node...

- Inserting a node with values 55



Deleting a Node.

- The delete function deletes a node from the binary search tree
  - In order to take care the properties of binary search tree, we can divide the deleting functions into three categories
    - Deleting a node that has no children
    - Deleting a node with one child
    - Deleing a node with two children



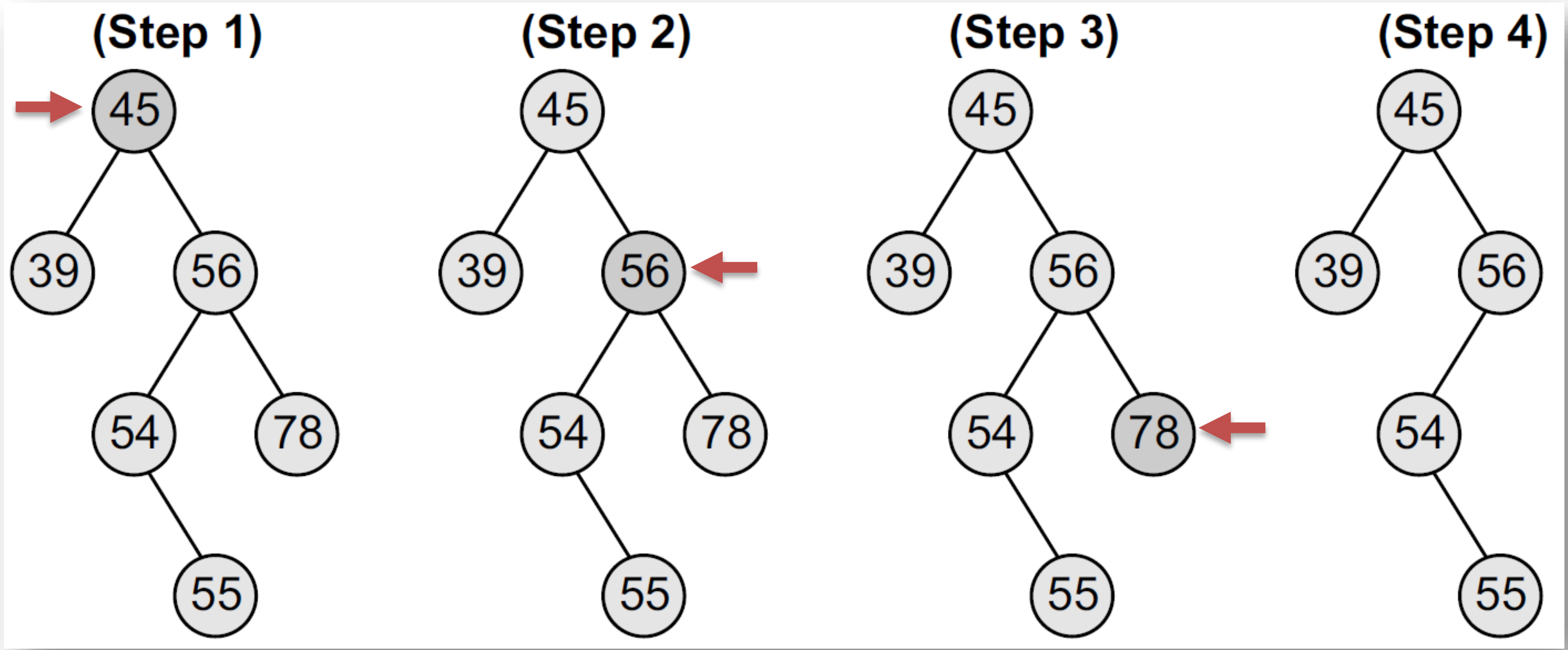
```
Delete (TREE, VAL)

Step 1: IF TREE = NULL
          Write "VAL not found in the tree"
        ELSE IF VAL < TREE -> DATA
          Delete(TREE->LEFT, VAL)
        ELSE IF VAL > TREE -> DATA
          Delete(TREE -> RIGHT, VAL)
        ELSE IF TREE -> LEFT AND TREE -> RIGHT
          SET TEMP = findLargestNode(TREE -> LEFT)
          SET TREE -> DATA = TEMP -> DATA
          Delete(TREE -> LEFT, TEMP -> DATA)
        ELSE
          SET TEMP = TREE
          IF TREE -> LEFT = NULL AND TREE -> RIGHT = NULL
              SET TREE = NULL
          ELSE IF TREE -> LEFT != NULL
               SET TREE = TREE -> LEFT
          ELSE
               SET TREE = TREE -> RIGHT
          [END OF IF]
          FREE TEMP
        [END OF IF]
Step 2: END
```

Deleting a Node..

- Deleting a node that has no children
  - Simply remove this node without any issue
  - The simplest case of deletion

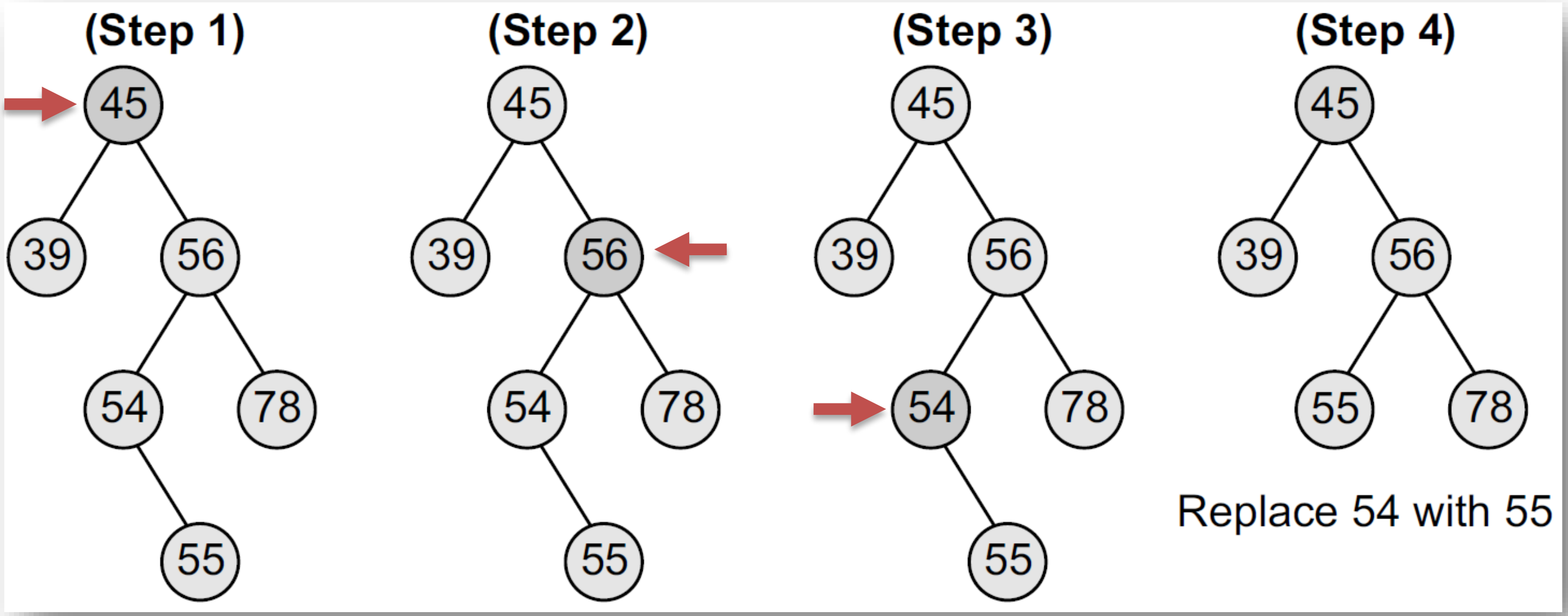
- Deleting node 78 from the given binary search tree



Deleting a Node...

- Deleting a node with one child
  - Replace the node with its child
  - It is also simple!

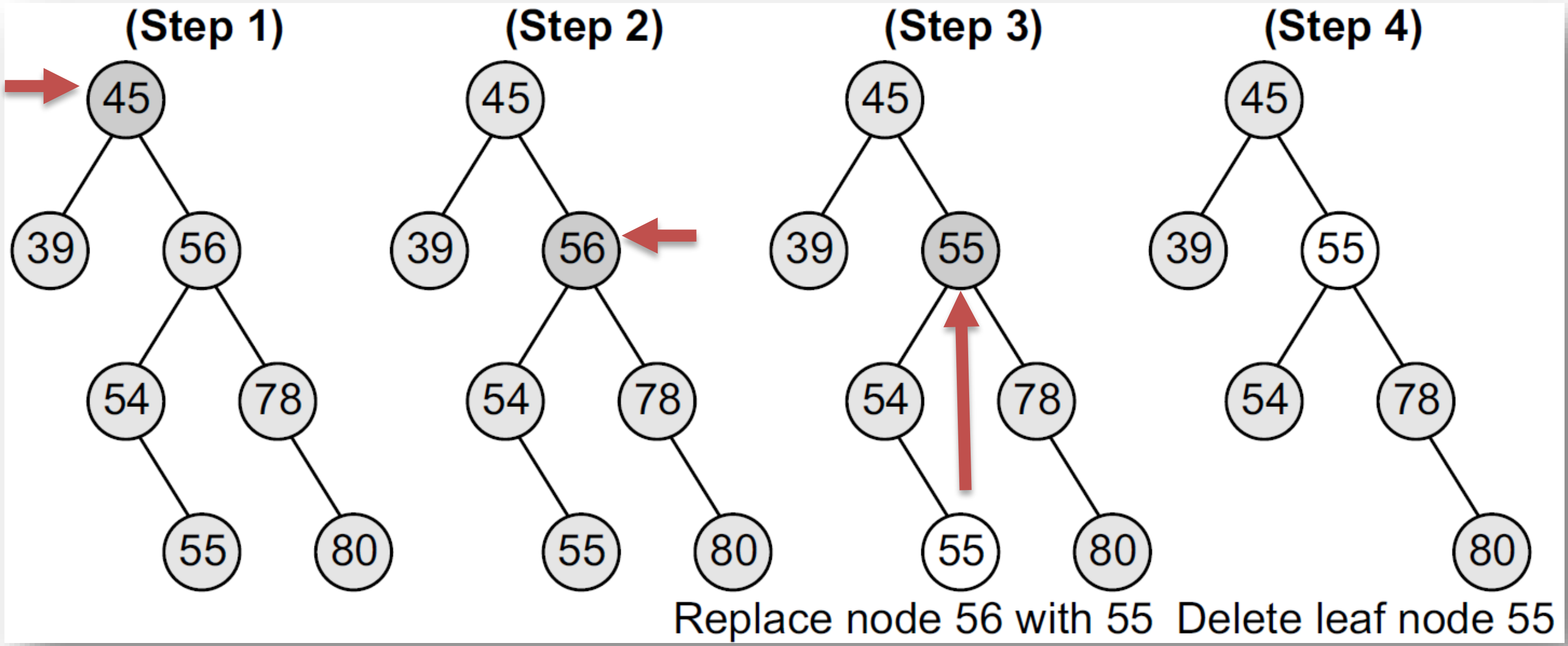
- Deleting node 54 from the given binary search tree



Deleting a Node….

- Deleing a node with two children
  - Replace the node's value with its **in-order predecessor** (largest value in the left sub-tree) or **in-order successor** (smallest value in the right sub-tree)

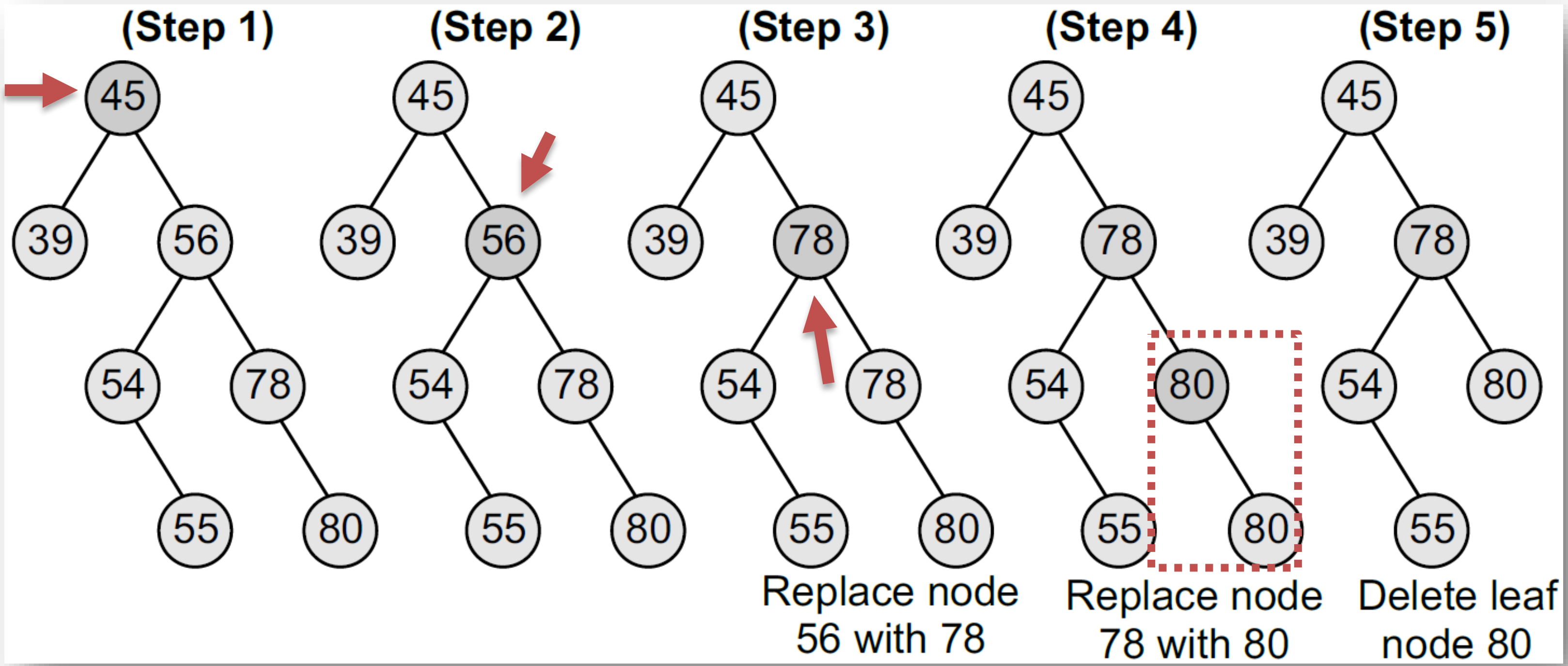
- Deleting node 56 from the given binary search tree



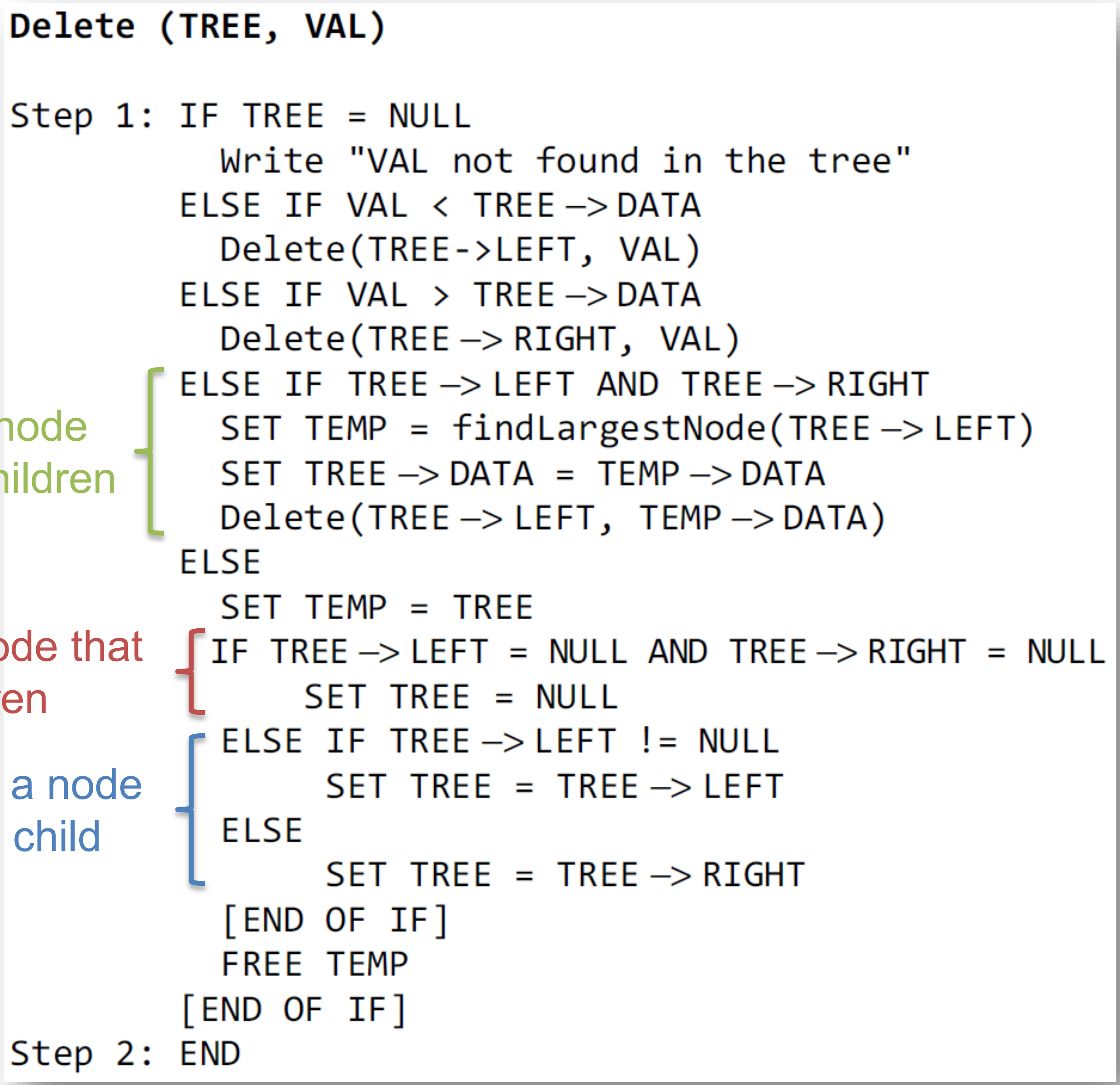
Deleting a Node.....

- Deleing a node with two children
  - Replace the node's value with its **in-order predecessor** (largest value in the left sub-tree) or **in-order successor** (smallest value in the right sub-tree)

- Deleting node 56 from the given binary search tree



Deleting a Node……



```
Delete (TREE, VAL)

Step 1: IF TREE = NULL
          Write "VAL not found in the tree"
        ELSE IF VAL < TREE -> DATA
          Delete(TREE->LEFT, VAL)
        ELSE IF VAL > TREE -> DATA
          Delete(TREE -> RIGHT, VAL)
        ELSE IF TREE -> LEFT AND TREE -> RIGHT
          SET TEMP = findLargestNode(TREE -> LEFT)
          SET TREE -> DATA = TEMP -> DATA
          Delete(TREE -> LEFT, TEMP -> DATA)
        ELSE
          SET TEMP = TREE
          IF TREE -> LEFT = NULL AND TREE -> RIGHT = NULL
              SET TREE = NULL
          ELSE IF TREE -> LEFT != NULL
              SET TREE = TREE -> LEFT
          ELSE
              SET TREE = TREE -> RIGHT
          [END OF IF]
          FREE TEMP
        [END OF IF]
Step 2: END
```

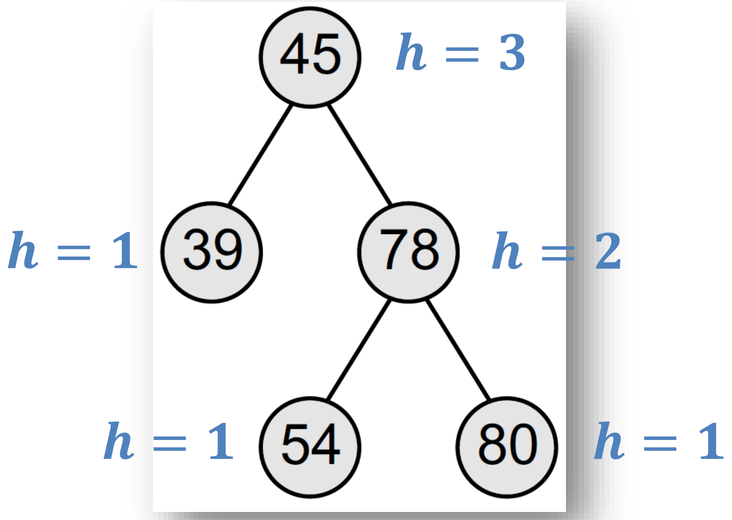
Deleing a node with two children

Deleting a node that has no children

Deleting a node with one child

Height of a Node

- Height for a node in a binary search tree
  - In order to determine the height of a node in a binary search tree, we calculate the height of its left sub-tree h_L and the right sub-tree h_R
  - After that, the height of the node is $1 + \max(h_L, h_R)$

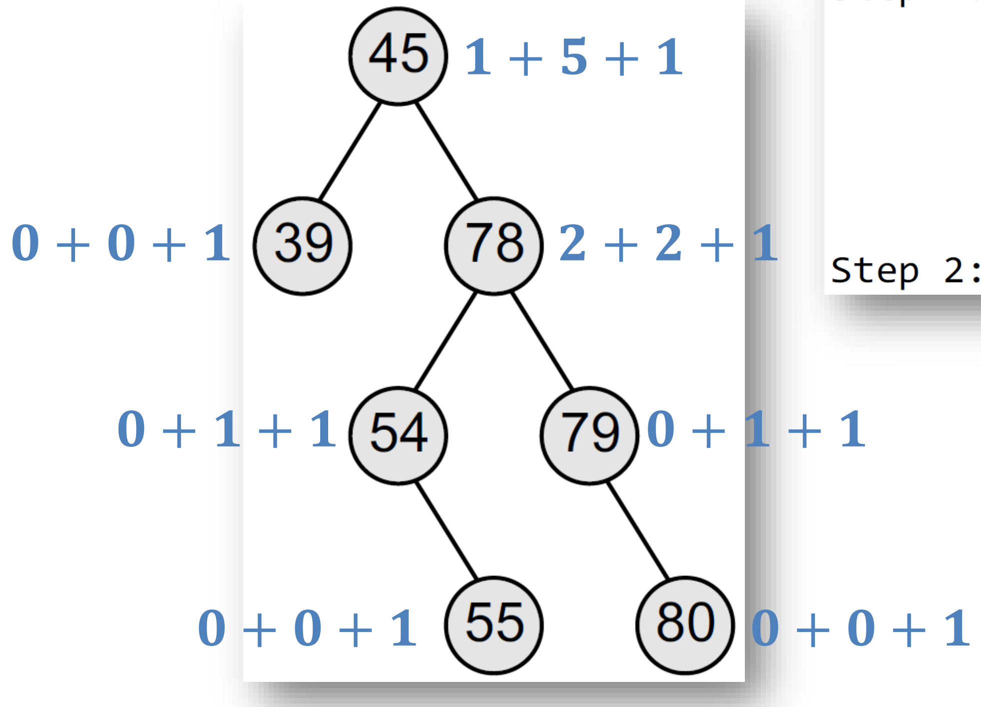


```
Height (TREE)

Step 1: IF TREE = NULL
             Return 0
        ELSE
         SET LeftHeight = Height(TREE —> LEFT)
         SET RightHeight = Height(TREE —> RIGHT)
          IF LeftHeight > RightHeight
               Return LeftHeight + 1
          ELSE
               Return RightHeight + 1
          [END OF IF]
        [END OF IF]
Step 2: END
```

Number of Nodes in a Binary Search Tree

- Determining the number of nodes in a binary search tree is similar to determining its height
  - Number of nodes in a binary search tree is the sum of number of nodes in left sub-tree, right sub-tree and 1



```
totalNodes(TREE)

Step 1: IF TREE = NULL
              Return 0
        ELSE
              Return totalNodes(TREE -> LEFT)
                   + totalNodes(TREE -> RIGHT) + 1
        [END OF IF]
Step 2: END
```

Number of External Nodes

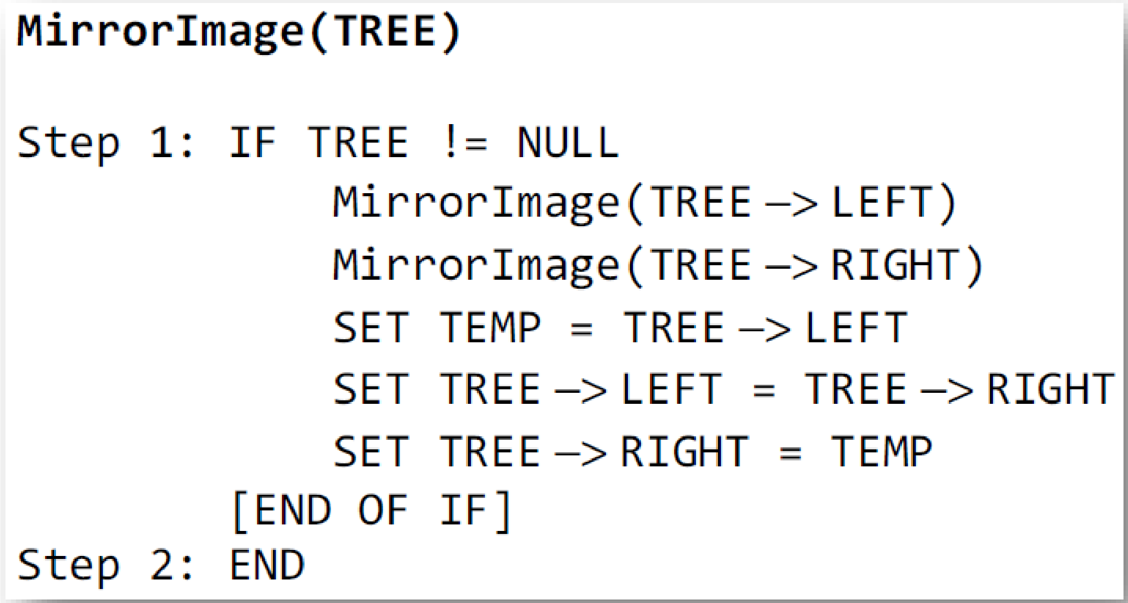
- The total number of external nodes or leaf nodes can be calculated by adding the number of external nodes in the left sub-tree and the right sub-tree
  - If the tree is empty, then the number of external nodes will be zero
  - If there is only one node in the tree, then the number of external nodes will be one
  - The internal nodes can be obtained by using the total nodes and total external nodes in the binary search tree

```
totalExternalNodes(TREE)

Step 1: IF TREE = NULL
             Return 0
          ELSE IF TREE -> LEFT = NULL AND TREE -> RIGHT = NULL
             Return 1
        ELSE
             Return totalExternalNodes(TREE -> LEFT) +
             totalExternalNodes(TREE -> RIGHT)
        [END OF IF]
Step 2: END
```

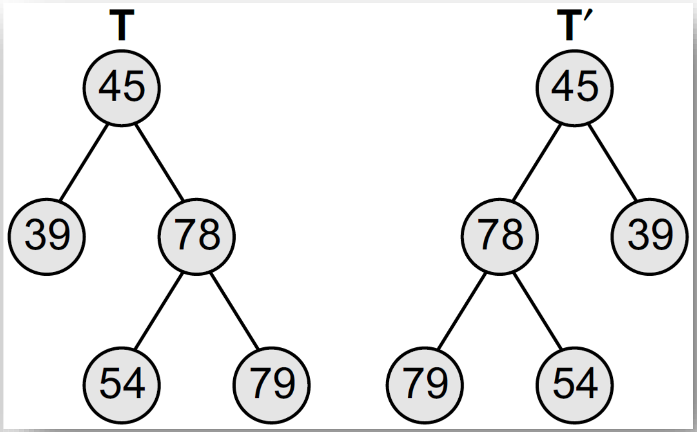
Mirror of a Binary Search Tree

- Mirror image of a binary search tree is obtained by interchanging the left sub-tree with the right sub-tree at every node of the tree



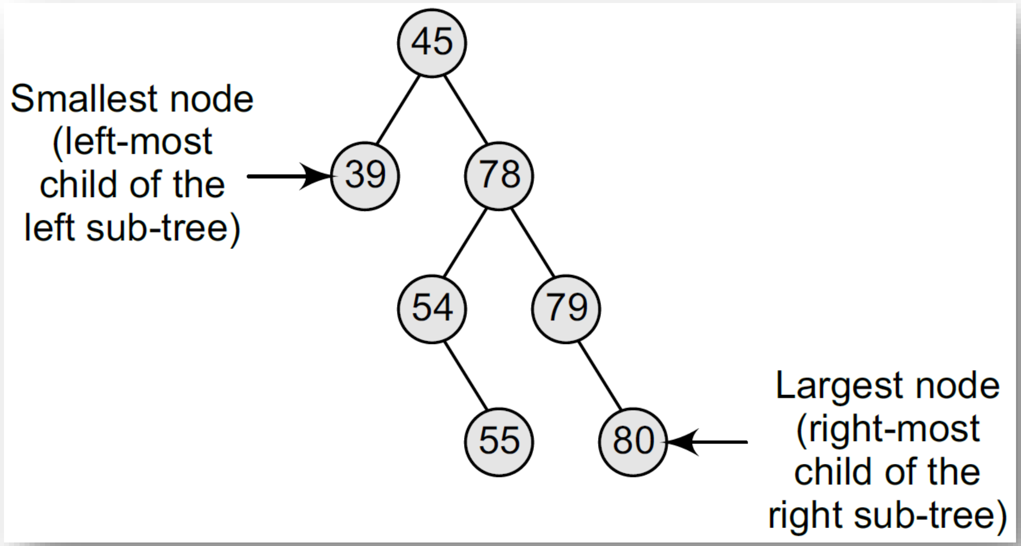
```
MirrorImage(TREE)

Step 1: IF TREE != NULL
                MirrorImage(TREE -> LEFT)
                MirrorImage(TREE -> RIGHT)
                SET TEMP = TREE -> LEFT
                SET TREE -> LEFT = TREE -> RIGHT
                SET TREE -> RIGHT = TEMP
            [END OF IF]
Step 2: END
```



Finding the Smallest/Largest Node

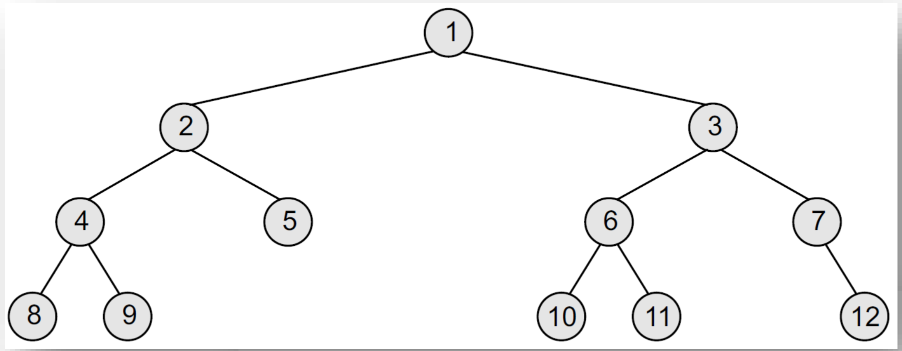
- The very basic property of the binary search tree states that the smaller value will occur in the left sub-tree
  - If the left sub-tree is NULL, then the value of the root node will be smallest

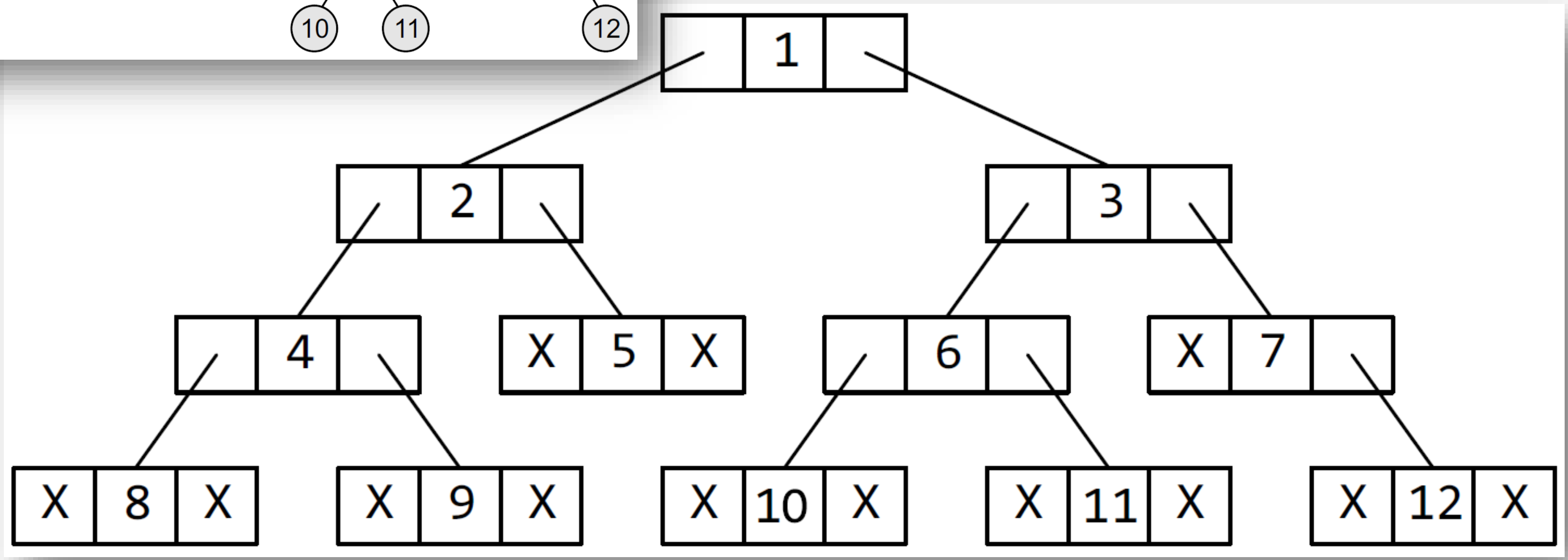

45
39
78
54
79
55
80
Smallest node (left-most child of the left sub-tree)
Largest node (right-most child of the right sub-tree)

- To find the node with the largest value, we find the value of the rightmost node of the right subtree
  - If the right sub-tree is empty, then the root node will be the largest value in the tree

Threaded Binary Trees

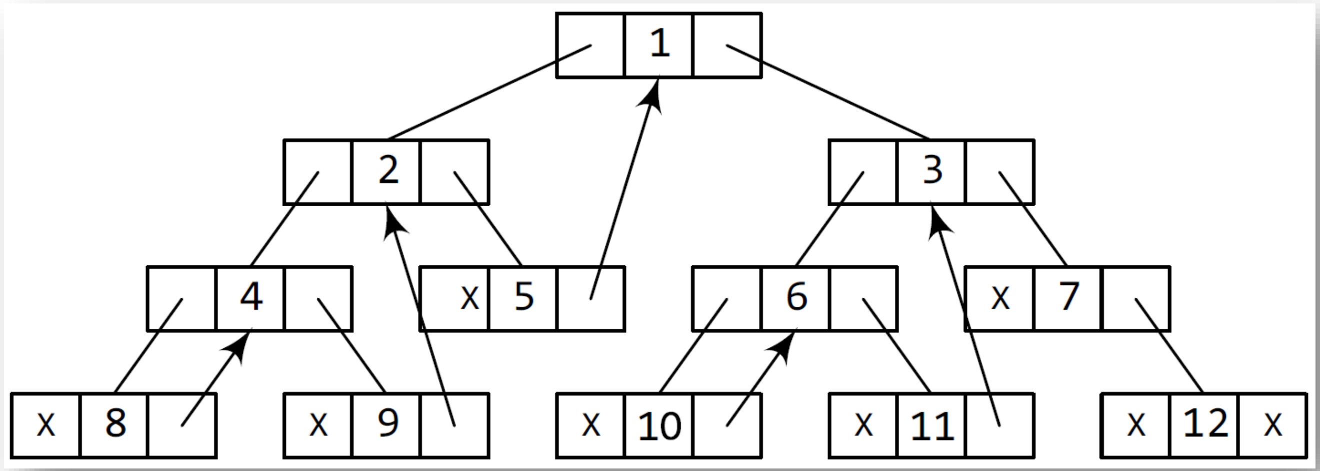
- A threaded binary tree is the same as that of a binary tree but with a difference in storing the NULL pointers
  - The space that is wasted in storing a NULL pointer can be efficiently used to store some other useful piece of information





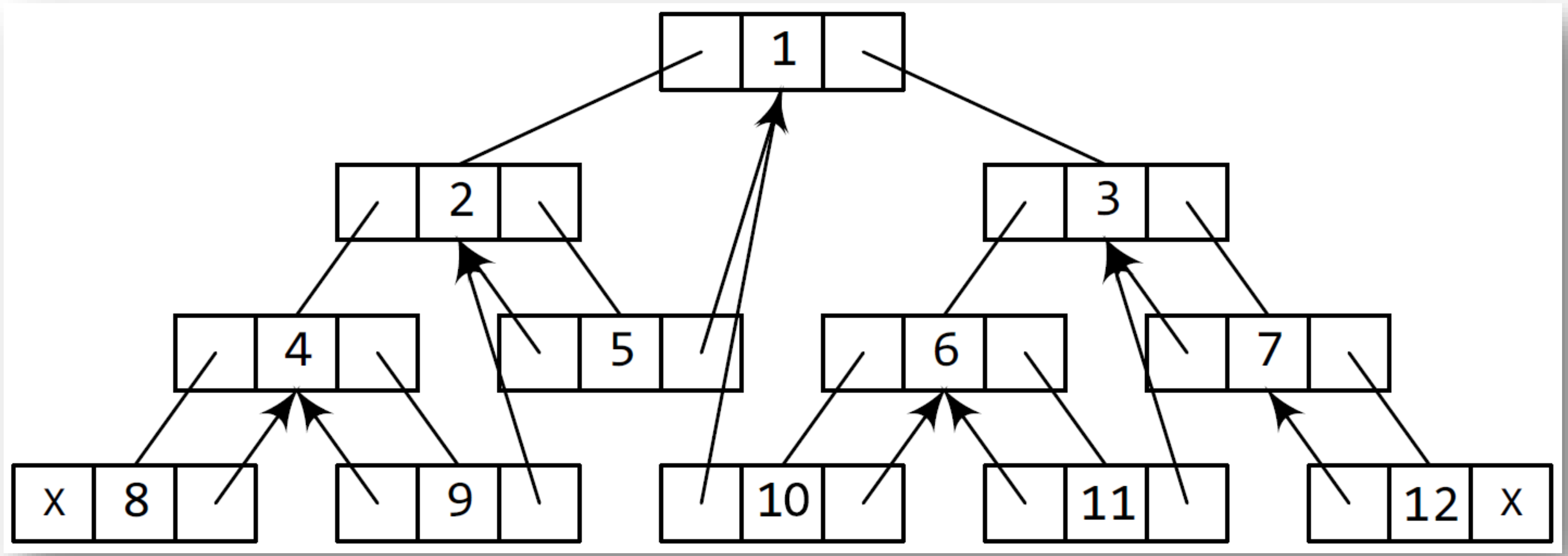
One-way Threaded Trees

- A **one-way threaded tree** is also called a **single-threaded tree**
  - If the thread appears in the **left field**, then the left field will be made to point to the **in-order predecessor** of the node
    - Such a tree is called a **left-threaded binary tree**
  - If the thread appears in the **right field**, then it will point to the **in-order successor** of the node
    - Such a tree is called a **right-threaded binary tree**



Two-way Threaded Trees

- In a two-way threaded tree, also called a **double-threaded tree**, threads will appear in both the left and the right field of the node
  - The left field will point to the in-order predecessor of the node, and the right field will point to its successor
  - A two-way threaded binary tree is also called a **fully threaded binary tree**



Questions?

kychen@mail.ntust.edu.tw