بِسْمِ ٱللَّهِ ٱلرَّحْمَٰنِ ٱلرَّحِيمِ

**In the name of Allah, Most Gracious, Most Merciful**

# CSE 4303
# Data Structure

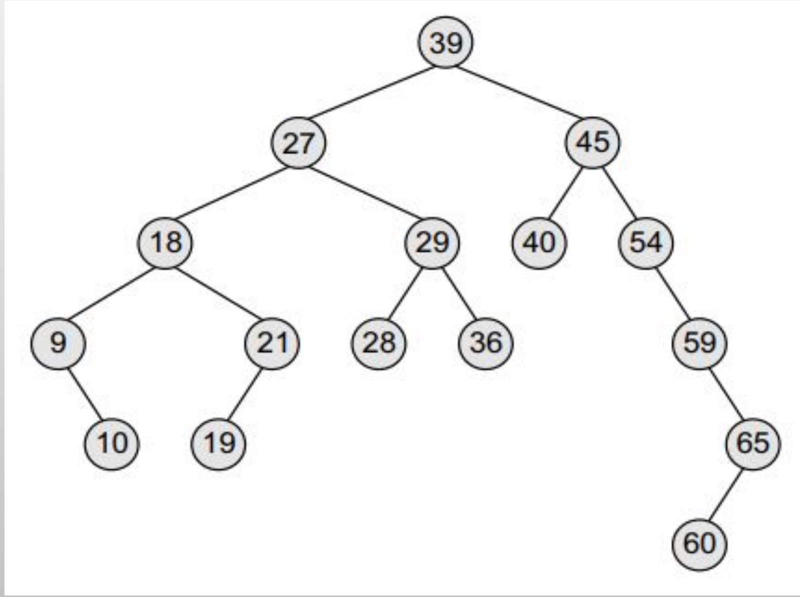
Topic: Binary Search Tree

Asaduzzaman Herok
Lecturer | CSE | IUT
asaduzzaman34@iut-dhaka.edu

# BINARY SEARCH TREES (BST)

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.

- The left sub-tree of a node N contains values that are less than N's value.
- The right sub-tree of a node N contains values that are greater than N's value.
- Both the left and the right binary trees also satisfy these properties and, thus, are 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.

# WHY DO WE NEED BST

Binary Search on sorted Array:

- Time complexity O(log(n)).
- Insertion of new element ?
  - Need to sort again
- Deletion of element?
  - Need to sort again

Binary Search Tree:

- Average time complexity O(log(n)).
- Insertion of new element ?
  - Still O(log(n))
- Deletion of element?
  - Still O(log(n))

Worst Case? …. O(n) 😡

## Some Uses

- BSTs are used for indexing and multi-level indexing.
- They are also helpful to implement various searching algorithms.
- It is helpful in maintaining a sorted stream of data.
- TreeMap and TreeSet data structures are internally implemented using self-balancing BSTs
- BSTs are widely used in dictionary problems

# OPERATIONS ON BST

- Searching an element
- Inserting a new element
- Deleting an element
- Deleting the entire tree
- Determining the height of the tree
- Finding the largest element
- Finding the smallest element
- Traversals (Pre-Order, In-Order, Post-Order)

# SEARCHING IN BST

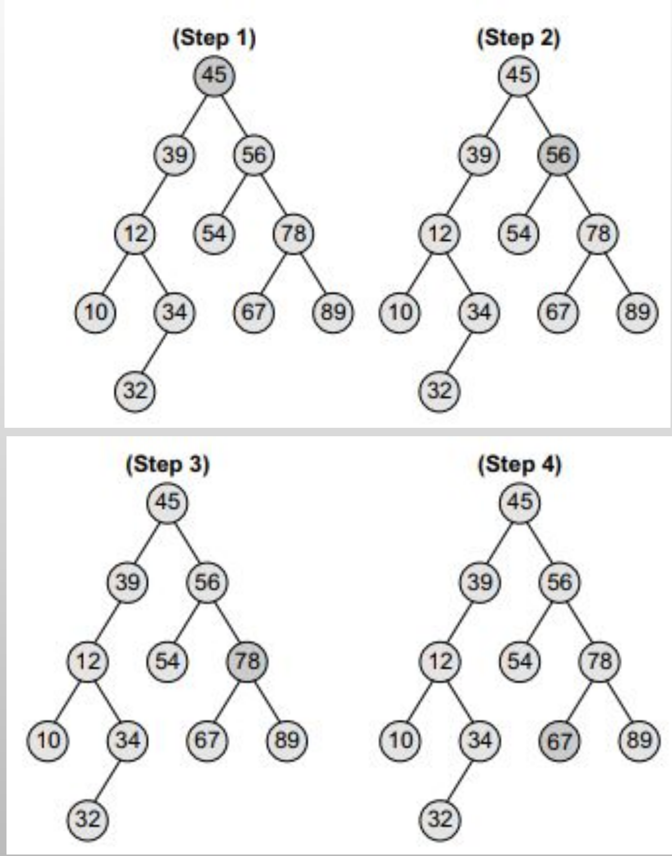



**Figure 10.6** Searching a node with value 67 in the given binary search tree

```
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
```

# INSERTION IN BST

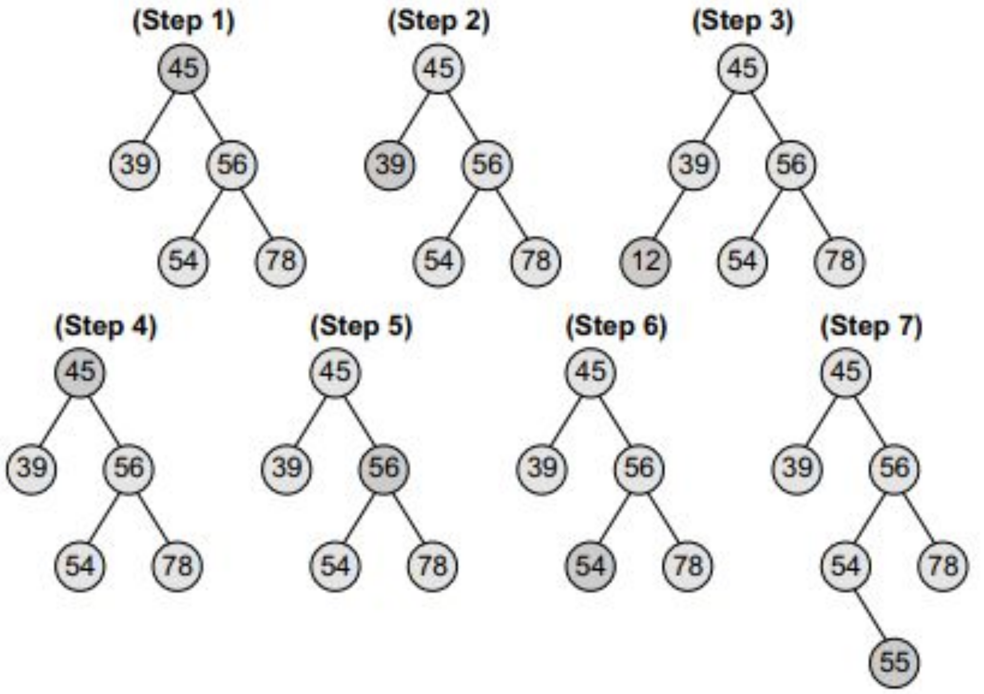



**Figure 10.10** Inserting nodes with values 12 and 55 in the given binary search tree

```
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
```

# DELETION IN BST

Case 1: Deleting a Node that has No Children

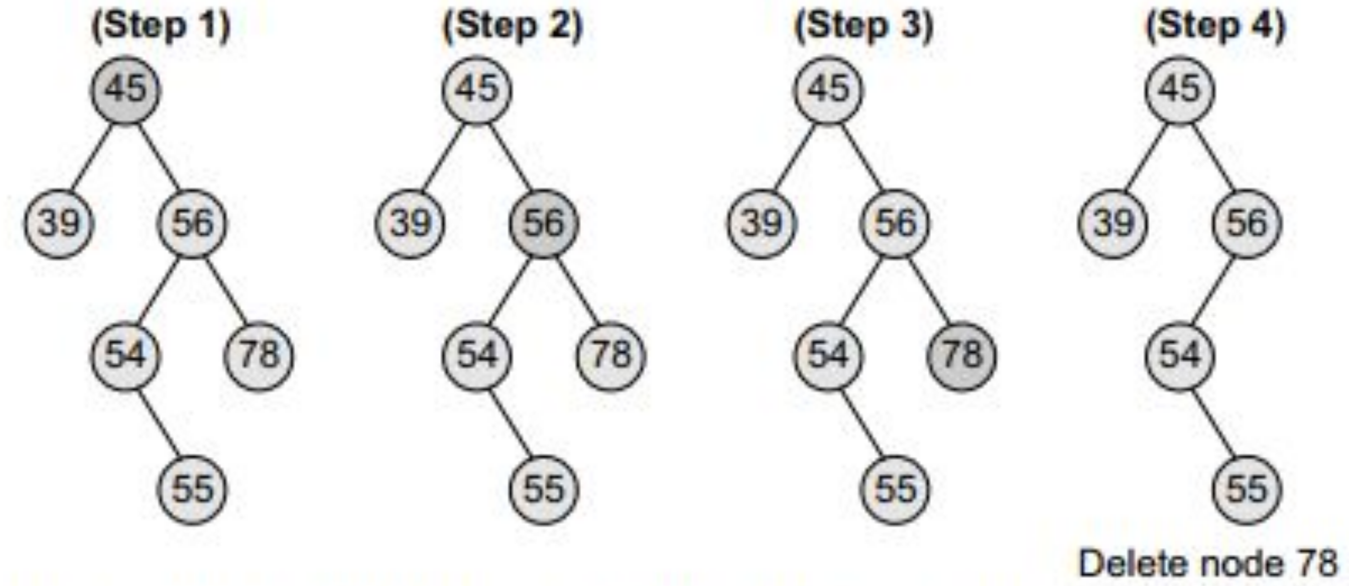



**Figure 10.11** Deleting node 78 from the given binary search tree

# DELETION IN BST

Case 2: Deleting a Node with One Child

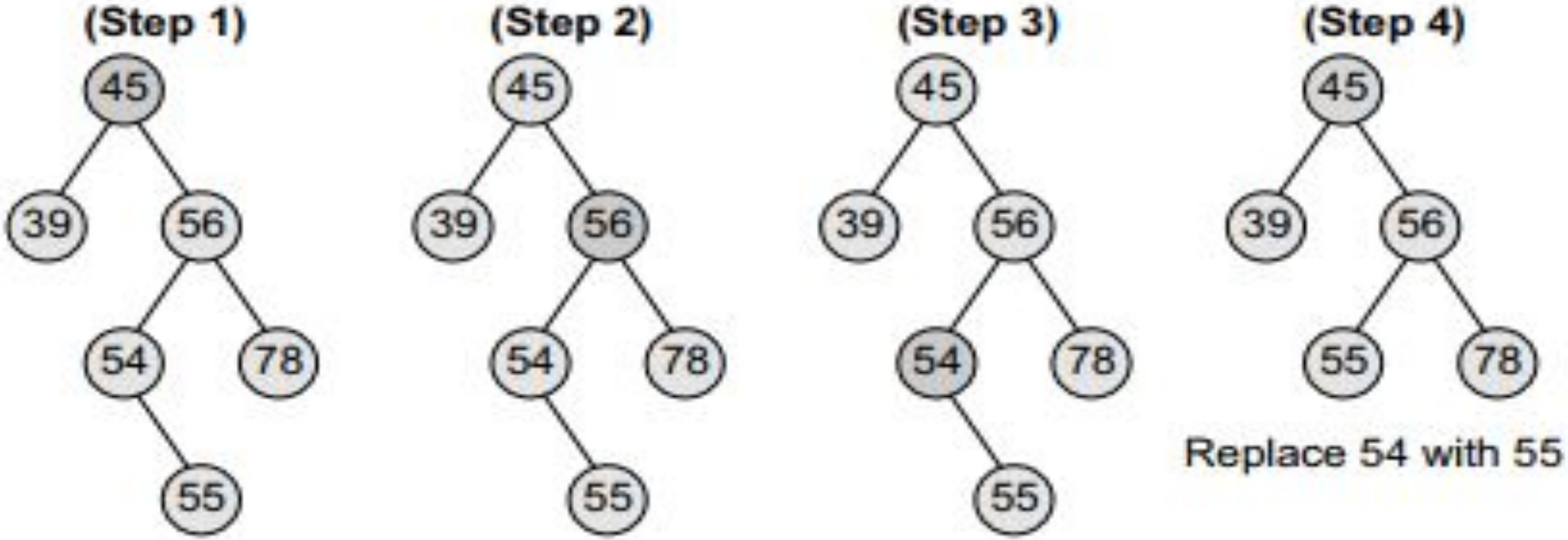



**Figure 10.12** Deleting node 54 from the given binary search tree

# DELETION IN BST

Case 3: Deleting a Node with Two Children

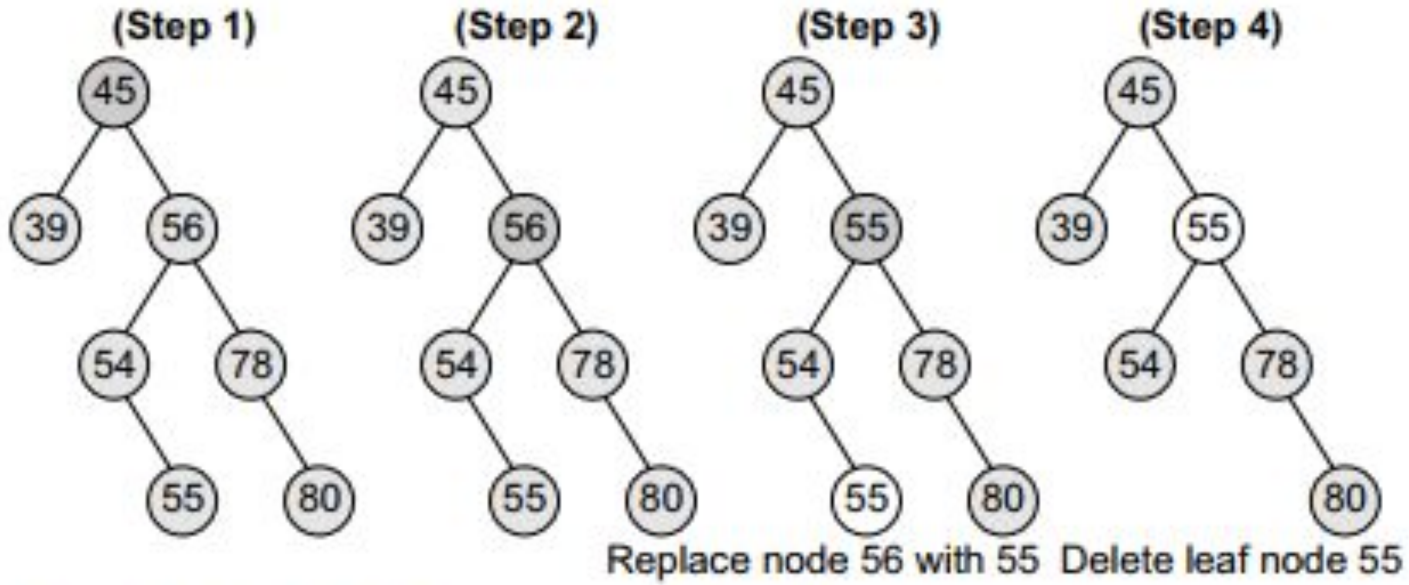



Figure 10.13 Deleting node 56 from the given binary search tree

# DELETION IN BST

## The Pseudocode

```
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
```

## DELETION OF ENTIRE BST

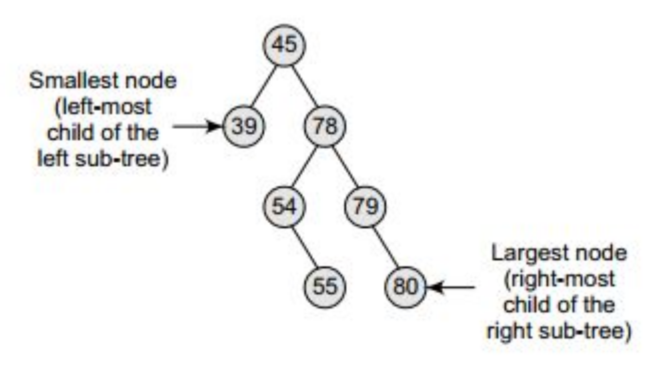
```
deleteTree(TREE)

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

## SMALLEST ELEMENT OF BST

```
findSmallestElement(TREE)

Step 1: IF TREE = NULL OR TREE -> LEFT = NULL
            Returen TREE
        ELSE
            Return findSmallestElement(TREE -> LEFT)
        [END OF IF]
Step 2: END
```

## LARGEST ELEMENT OF BST

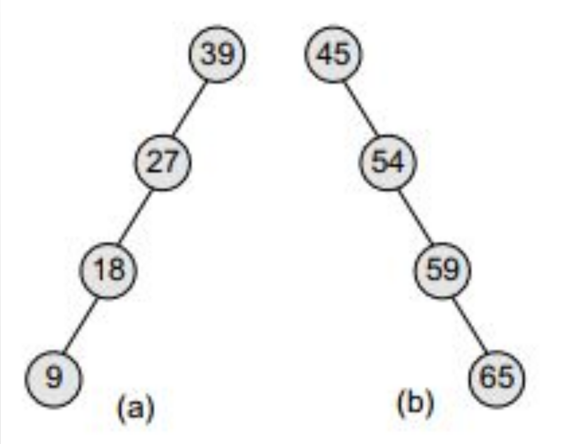
```
findLargestElement(TREE)

Step 1: IF TREE = NULL OR TREE -> RIGHT = NULL
            Return TREE
        ELSE
           Return findLargestElement(TREE -> RIGHT)
        [END OF IF]
Step 2: END
```



# DISADVANTAGES OF BST

Worst Case Scenario:

- Searching: O(n)
- Insertion: O(n)
- Deletion: O(n)
- Height: n




(a) Left skewed, and (b) right skewed binary search trees


**Acknowledgements**

Data Structures
Using
c
Reema Thareja