Data Structures (15B11CI311)

Odd Semester 2020

3rd Semester , Computer Science and Engineering

Jaypee Institute Of Information Technology (JIIT), Noida

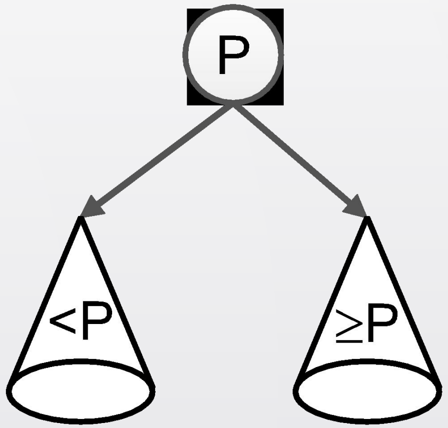
Lecture: 28

Topics to be covered:

- Introduction to BST
- Operations on BST

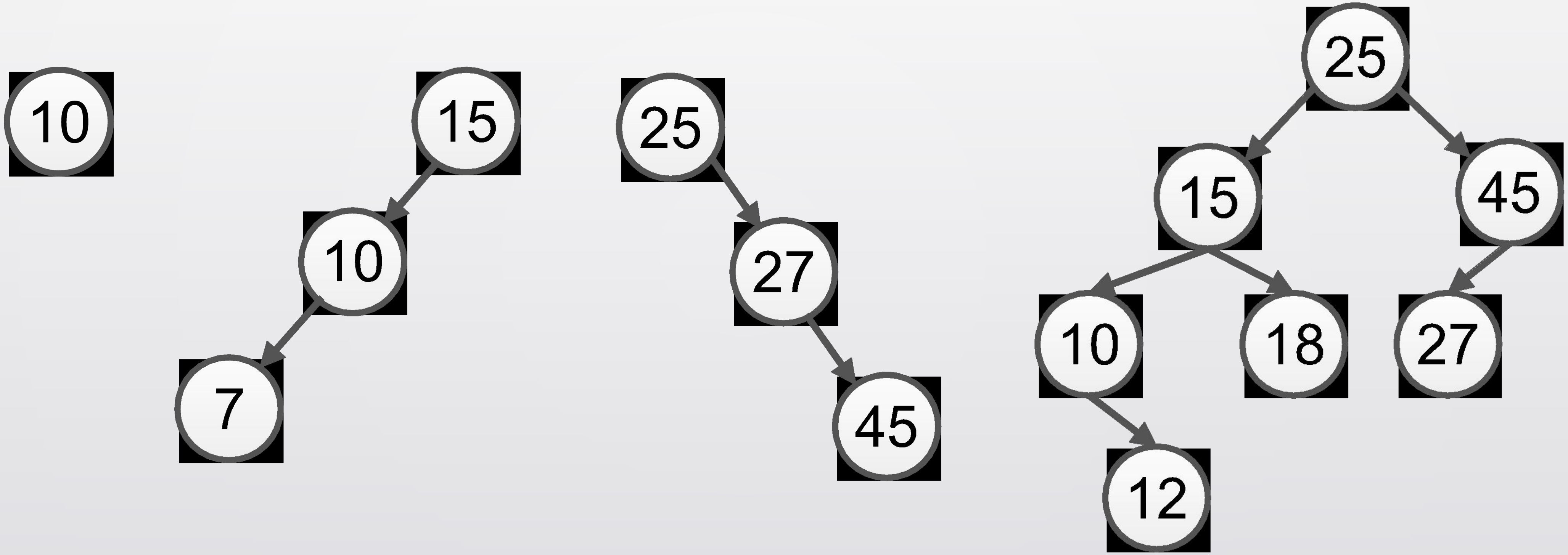
Binary Search Tree

- A Binary Search Tree is a binary tree with the following properties:
  - The left sub-tree nodes contain less value than the root node value.
  - The right sub-tree nodes contain greater or equal value to the root node value.
  - Each sub-tree is itself a binary search tree.
- It is used to search the data efficiently along with insertion and deletion.



Binary Search Tree

- Some of the Valid BST



BST Operations

- Traversals:
  - Like binary tree Inorder, Preorder and post order traversals
- Searches
- Insertion
- Deletion

Search a node in a BST

- Smallest node
- Largest node
- A requested node

Find the smallest node


```
Algorithm findSmallestBST (root)
This algorithm finds the smallest node in a BST.
   Pre    root is a pointer to a nonempty BST or subtree
   Return address of smallest node
1  if (left subtree empty)
   1  return (root)
2  end if
3  return findSmallestBST (left subtree)
end findSmallestBST
```

```c
int minValue(struct node* node)
{
struct node* current = node;

/* loop down to find the leftmost leaf */
while (current->left != NULL)
{
    current = current->left;
}
return(current->data);
}
```

Source: Data Structures A Pseudocode Approach with C, Second Edition by Richard F. Gilberg Behrouz A. Forouzan

Find the Largest node


```
Algorithm findLargestBST (root)
This algorithm finds the largest node in a BST.
   Pre     root is a pointer to a nonempty BST or subtree
   Return address of largest node returned
1  if (right subtree empty)
   1  return (root)
2  end if
3  return findLargestBST (right subtree)
end findLargestBST
```

```c
int maxValue(struct node* node)
{
    /* loop down to find the rightmost leaf */
    struct node* current = node;
    while (current->right != NULL)
        current = current->right;

    return (current->data);
}
```

Source: Data Structures A Pseudocode Approach with C, Second Edition by Richard F. Gilberg Behrouz A. Forouzan

Find the Requested node


```
Algorithm searchBST (root, targetKey)
Search a binary search tree for a given value.
   Pre     root is the root to a binary tree or subtree
           targetKey is the key value requested
   Return the node address if the value is found
           null if the node is not in the tree
1  if (empty tree)
      Not found
   1  return null
2  end if
3  if (targetKey < root)
   1  return searchBST (left subtree, targetKey)
4  else if (targetKey > root)
   1  return searchBST (right subtree, targetKey)
5  else
      Found target key
   1  return root
6  end if
end searchBST
```

```c
struct node* search(struct node* root, int key)
{
    // Base Cases: root is null or key is present at root
    if (root == NULL || root->key == key)
       return root;

    // Key is greater than root's key
    if (root->key < key)
       return search(root->right, key);

    // Key is smaller than root's key
    return search(root->left, key);
}
```

Searching

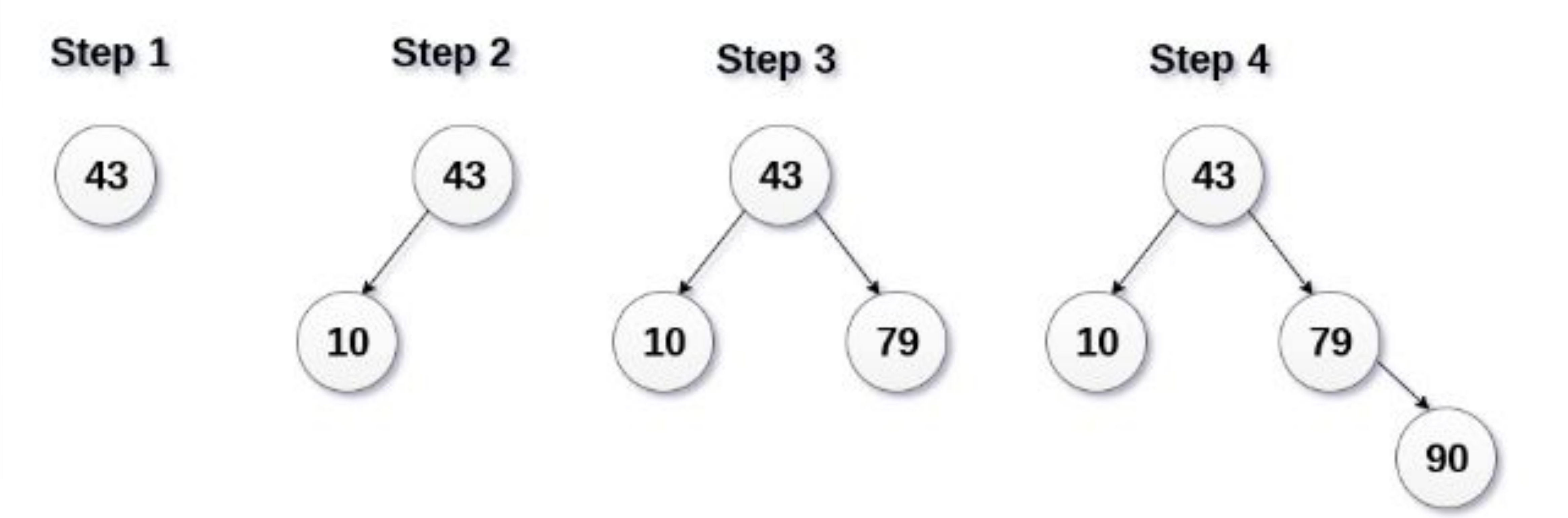
- Searching in a BST has O(h) worst-case runtime complexity, where h is the height of the tree.
- Minimum number of levels in a n node BST: O(log n)
  - It takes at least O(log n) comparisons to find a particular node.
- In worst case: O(n)
  - When the tree is right or left skewed

Insert into BST

- To insert new node, follow the branches to an empty sub-tree and then insert the new node.
- Insertion takes place at a leaf or a leaf like node: a node that has only one null subtree

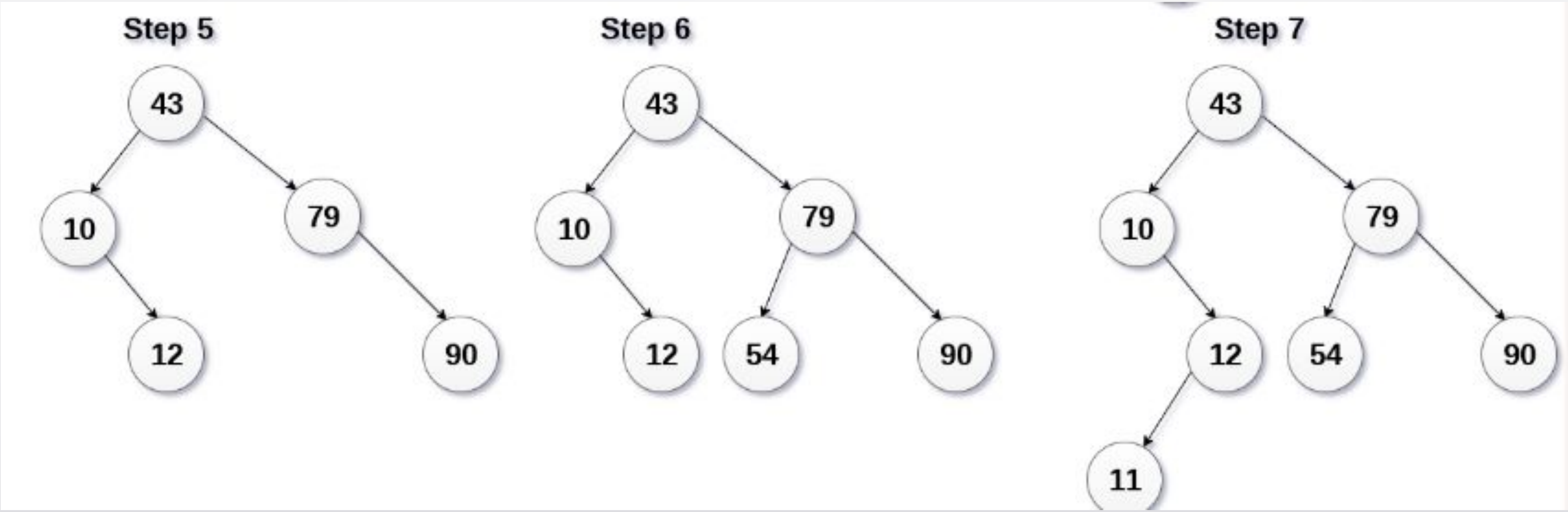
Insert into BST

- Elements to be inserted:
  - **43, 10, 79, 90, 12, 54, 11, 9, 50**



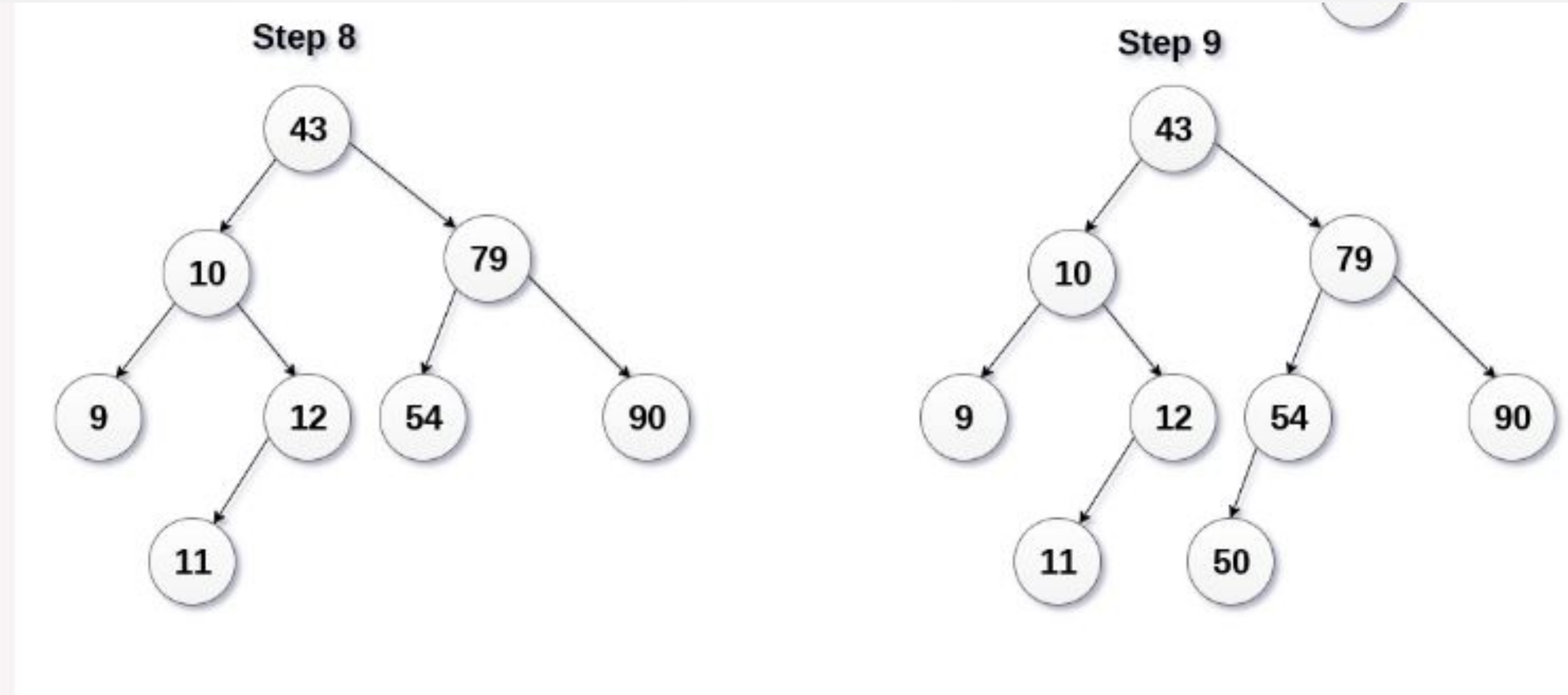
Insert into BST

- Elements to be inserted:
  - **43, 10, 79, 90, 12, 54, 11, 9, 50**



Insert into BST

- Elements to be inserted:
  - **43, 10, 79, 90, 12, 54, 11, 9, 50**



Insert into BST


```
Algorithm addBST (root, newNode)
Insert node containing new data into BST using recursion.
   Pre    root is address of current node in a BST
          newNode is address of node containing data
   Post   newNode inserted into the tree
   Return address of potential new tree root
1  if (empty tree)
   1  set root to newNode
   2  return newNode
2  end if
   Locate null subtree for insertion
3  if (newNode < root)
   1  return addBST (left subtree, newNode)
4  else
   1  return addBST (right subtree, newNode)
5  end if
end addBST
```

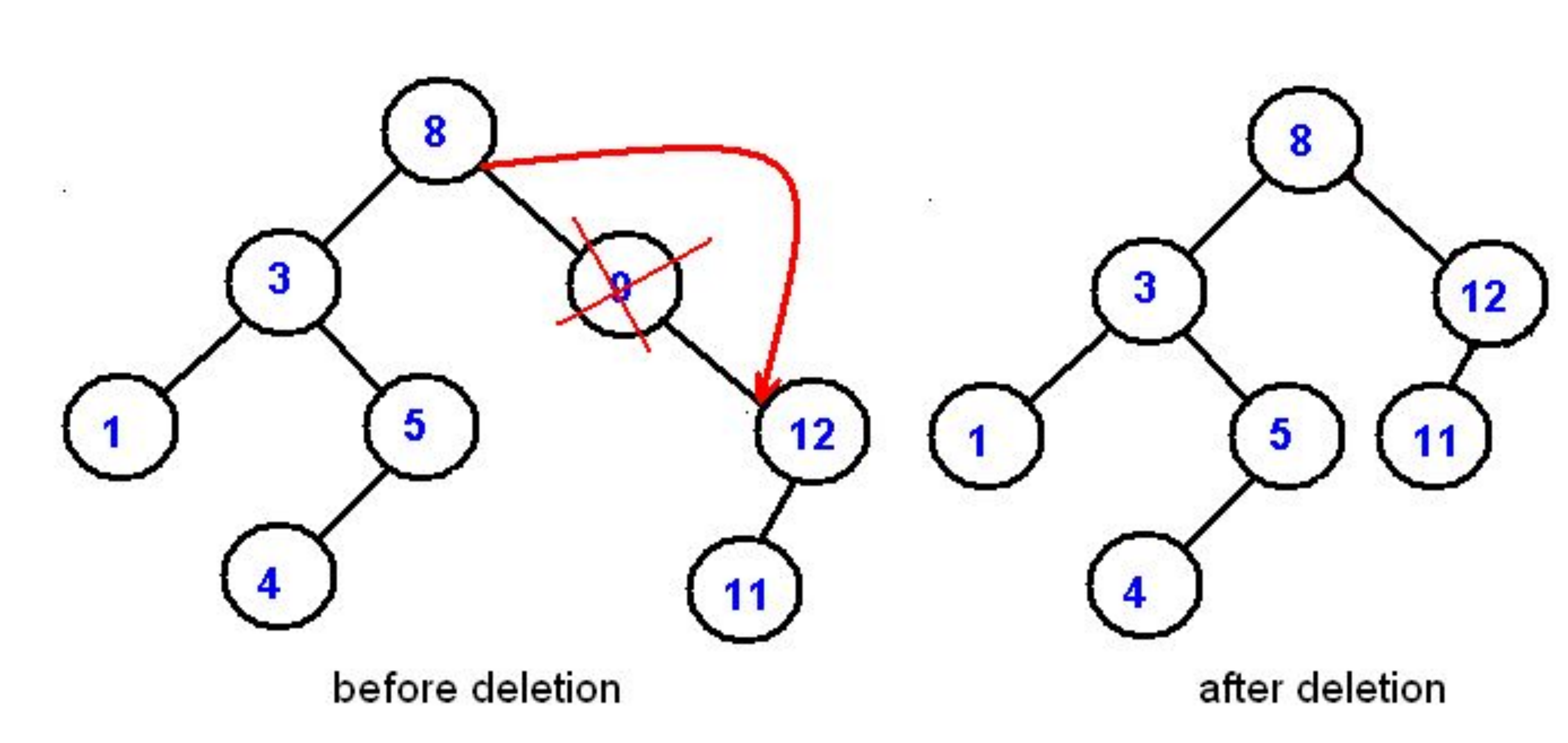
```c
struct node* insert(struct node* node, int key)
{
    /* If the tree is empty, return a new node */
    if (node == NULL)
        return newNode(key);

    /* Otherwise, recur down the tree */
    if (key < node->key)
        node->left = insert(node->left, key);
    else if (key > node->key)
        node->right = insert(node->right, key);

    /* return the (unchanged) node pointer */
    return node;
}
```

Source: Data Structures A Pseudocode Approach with C, Second Edition by Richard F. Gilberg Behrouz A. Forouzan

Deletion in BST

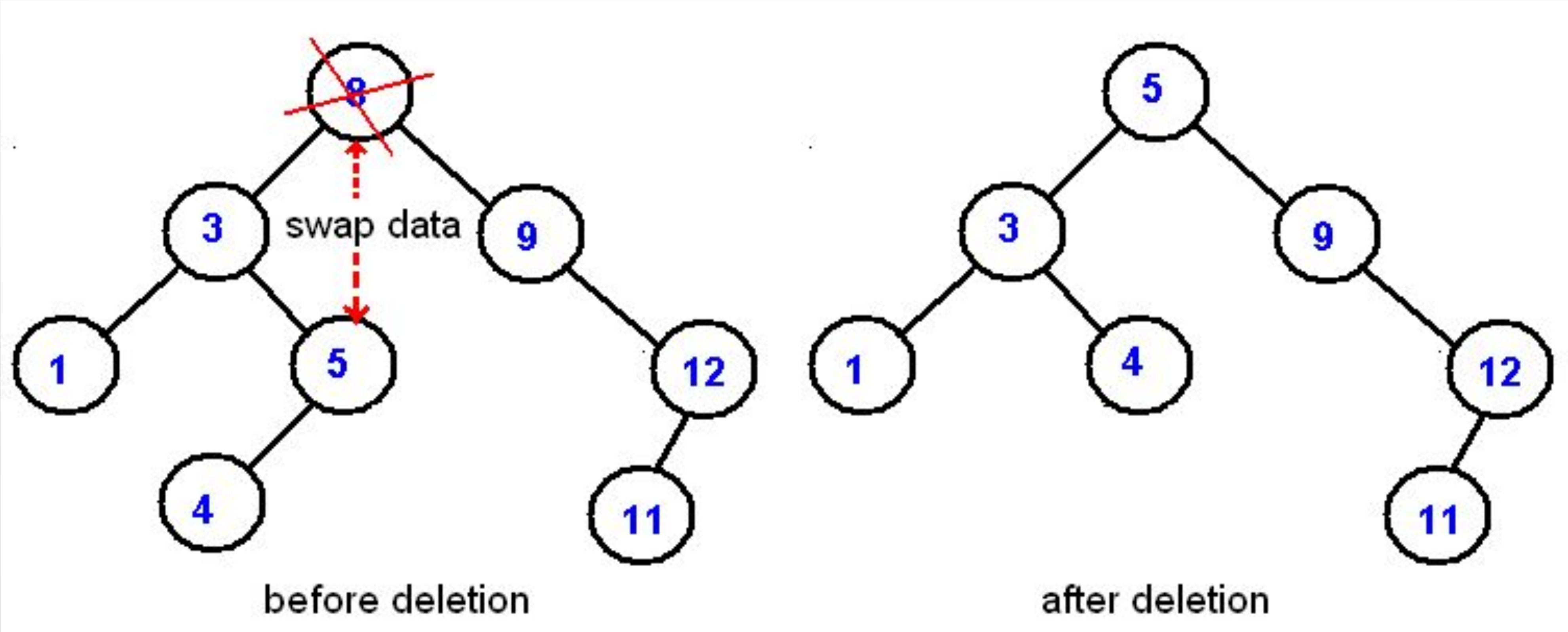
- Following are the cases to delete the node from a BST
  - The node which is to be deleted is not available in the tree;
  - The node which is to be deleted is a leaf node
    - Delete the node
- The node which is to be deleted has only one child
  - Delete the node and attach the subtree to the deleted node’s parent
- The node which is to be deleted has two children
  - Find the largest node in the deleted node’s left sub tree and move its data to replace the deleted node’s data. OR
  - Find the smallest node on the deleted node’s right sub tree and move its data to replace the deleted node’s data.
  - Either of these moves preserves the integrity of the binary search tree.

Deletion in BST



Deletion in BST

-



Deletion in BST

•

```
Algorithm deleteBST (root, dltKey)
This algorithm deletes a node from a BST.
   Pre     root is reference to node to be deleted
           dltKey is key of node to be deleted
   Post    node deleted
           if dltKey not found, root unchanged
   Return true if node deleted, false if not found
1 if (empty tree)
   1 return false
2 end if
3 if (dltKey < root)
   1 return deleteBST (left subtree, dltKey)
4 else if (dltKey > root)
   1 return deleteBST (right subtree, dltKey)
5 else
      Delete node found--test for leaf node
   1  If (no left subtree)
      1  make right subtree the root
      2  return true
   2  else if (no right subtree)
      1  make left subtree the root
      2  return true
   3  else
         Node to be deleted not a leaf. Find largest node on
         left subtree.
      1  save root in deleteNode
      2  set largest to largestBST (left subtree)
      3  move data in largest to deleteNode
      4  return deleteBST (left subtree of deleteNode,
                               key of largest
   4  end if
6 end if
end deleteBST
```

```c
struct node* deleteNode(struct node* root, int key)
{
    if (root == NULL) return root;
    // If the key to be deleted is smaller than the root's key, then it lies in left subtree
    if (key < root->key)
        root->left = deleteNode(root->left, key);
    // If the key to be deleted is greater than the root's key, then it lies in right subtree
    else if (key > root->key)
        root->right = deleteNode(root->right, key);
    // if key is same as root's key, then This is the node to be deleted
    else
    {   // node with only one child or no child
        if (root->left == NULL)
        {
            struct node *temp = root->right;
            delete(root);
            return temp;
        }
        else if (root->right == NULL)
        {
            struct node *temp = root->left;
            delete(root);
            return temp;
        }
        // node with two children: Get the inorder successor (smallest in the right subtree)
        struct node* temp = minValueNode(root->right);

        // Copy the inorder successor's content to this node
        root->key = temp->key;
        // Delete the inorder successor
        root->right = deleteNode(root->right, temp->key);
    }
    return root;
}
```

References

- *Cormen, Thomas H.; Leiserson, Charles E.; Rivest, Ronald L.; Stein, Clifford (2009) [1990]*
- *Introduction to Algorithms (3rd ed.). MIT Press and McGraw-Hill. ISBN 0-262-03384-4. 1320 pp.*
- Adam Drozdek, Data Structures and Algorithms in C++ (2nd Edition), 2001
- Data Structures A Pseudocode Approach with C, Second Edition by Richard F. Gilberg Behrouz A. Forouzan
- https://www.geeksforgeeks.org/iterative-postorder-traversal/
- https://www.tutorialspoint.com/cplusplus-program-to-perform-inorder-non-recursive-traversal-of-a-given-binary-tree
- https://www.tutorialspoint.com/cplusplus-program-to-perform-preorder-non-recursive-traversal-of-a-given-binary-tree
- **[1]https://www.cs.cmu.edu/~adamchik/15-121/lectures/Trees/trees.html**
- **[2]https://www.javatpoint.com/binary-search-tree**
- **[3] http://web.eecs.umich.edu/~akamil/teaching/su02/080802.ppt**