Binary Search Tree

Anoop S Babu

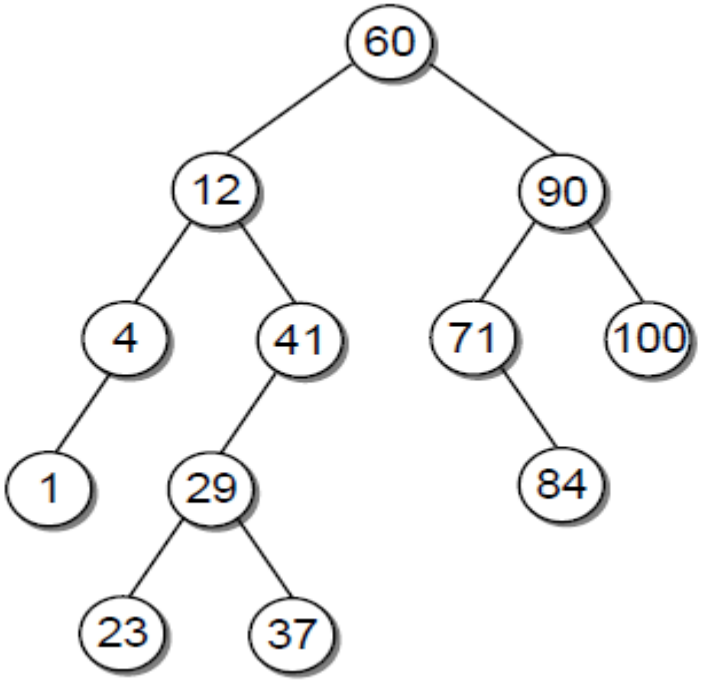
Faculty Associate

Dept. of Computer Science & Engineering

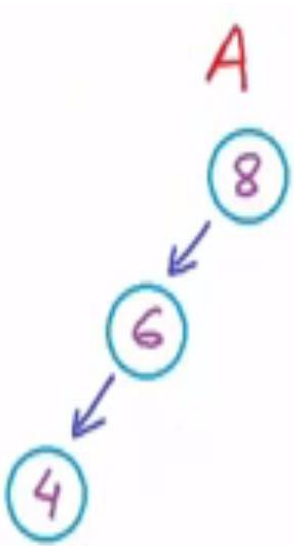
bsanoop@am.amrita.edu

Binary Search Tree (BST)

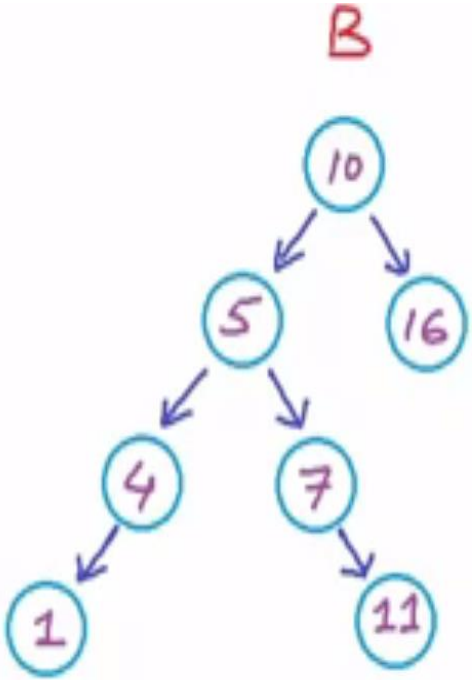
- A binary search tree (BST) is a binary tree in which,
  1. All nodes of left subtree are less than the root node.
  2. All nodes of right subtree are more than the root node.
  3. Both subtrees of each node are also BSTs



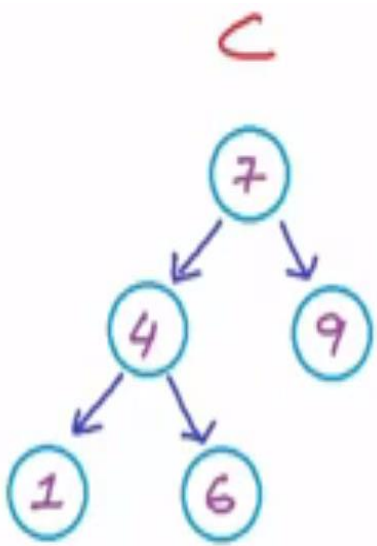
Binary Search Tree (BST)



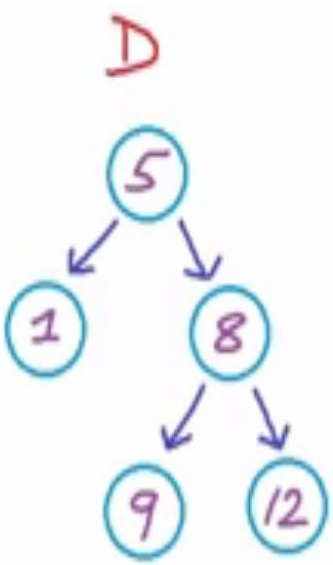
A is BST



B is not BST



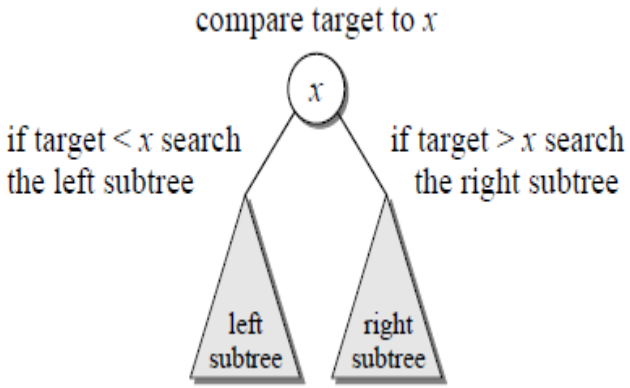
C is BST

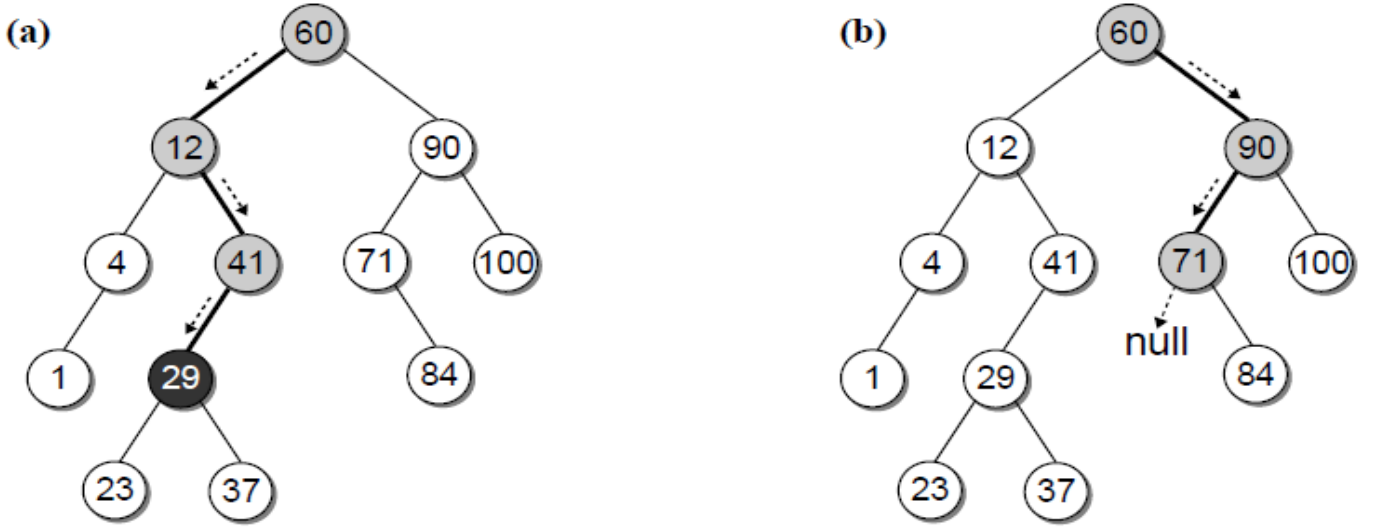


D is not BST

Searching in BST

- Whenever an element is to be searched,
  - Start searching from the root node.
  - Then if the data is less than the key value, search for the element in the left subtree.
  - Otherwise, search for the element in the right subtree.
  - Follow the same algorithm for each node.



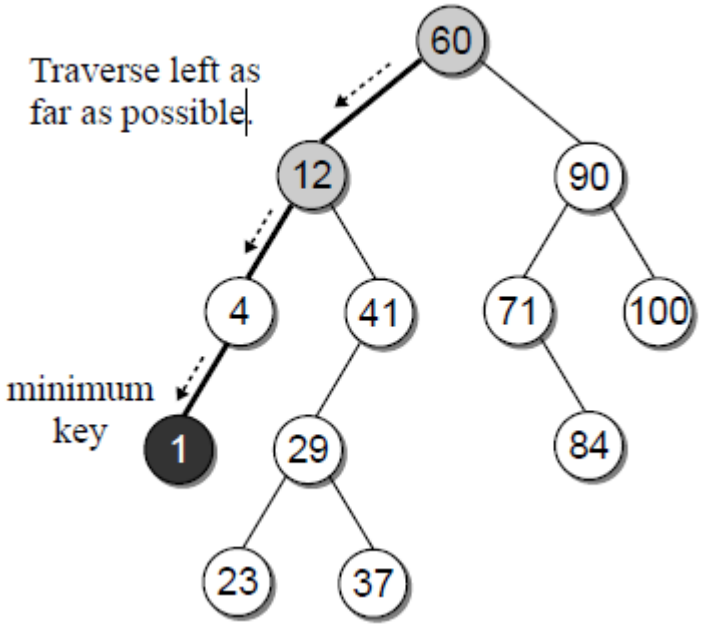


(a) successful search for 29 and (b) unsuccessful search for 68.

Min and Max Values in BST

- Finding the minimum value in BST: Traverse left as far as possible.

```
 # Helper method for finding the node containing the minimum key.
def _bstMinumum( self, subtree ):
  if subtree is None :
    return None
  elif subtree.left is None :
    return subtree
  else :
    return self._bstMinimum( subtree.left )
```

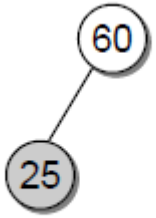


- Finding the maximum value in BST: Traverse right as far as possible.

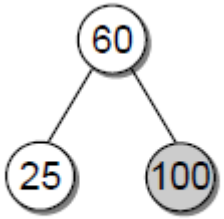
Constricting a BST

- Suppose we want to build a binary search tree from the key list [**60, 25, 100, 35, 17, 80**]

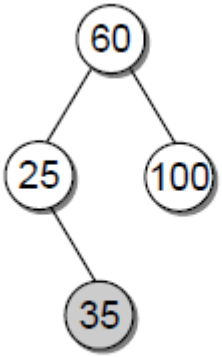
(a) Insert 60.



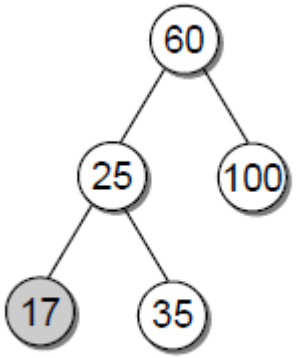
(b) Insert 25.



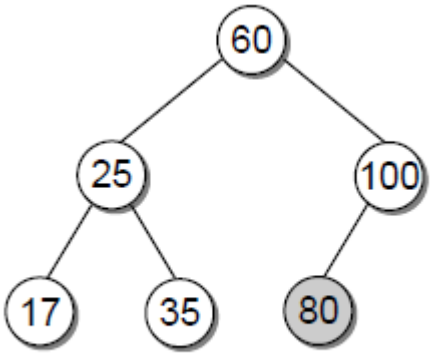
(c) Insert 100.



(d) Insert 35.



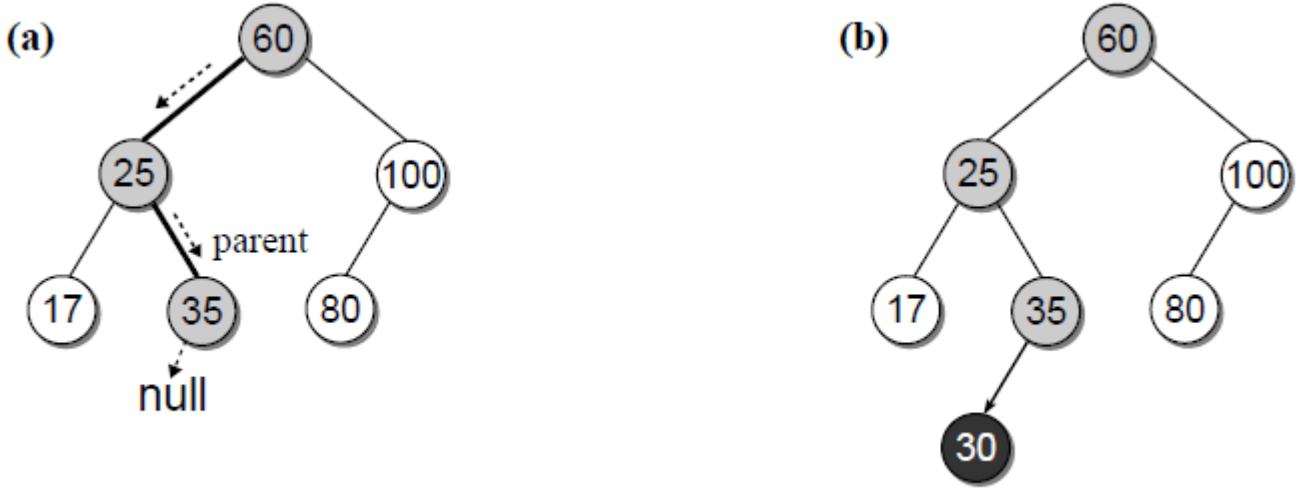
(e) Insert 17.



(f) Insert 80.

Insert Operation in BST

- Whenever an element is to be inserted,
  - First locate its proper location.
  - Start searching from the root node, then if the data is less than the key value, search for the empty location in the left subtree and insert the data.
  - Otherwise, search for the empty location in the right subtree and insert the data.



(a) searching for the node's location and (b) linking the new node into the tree.

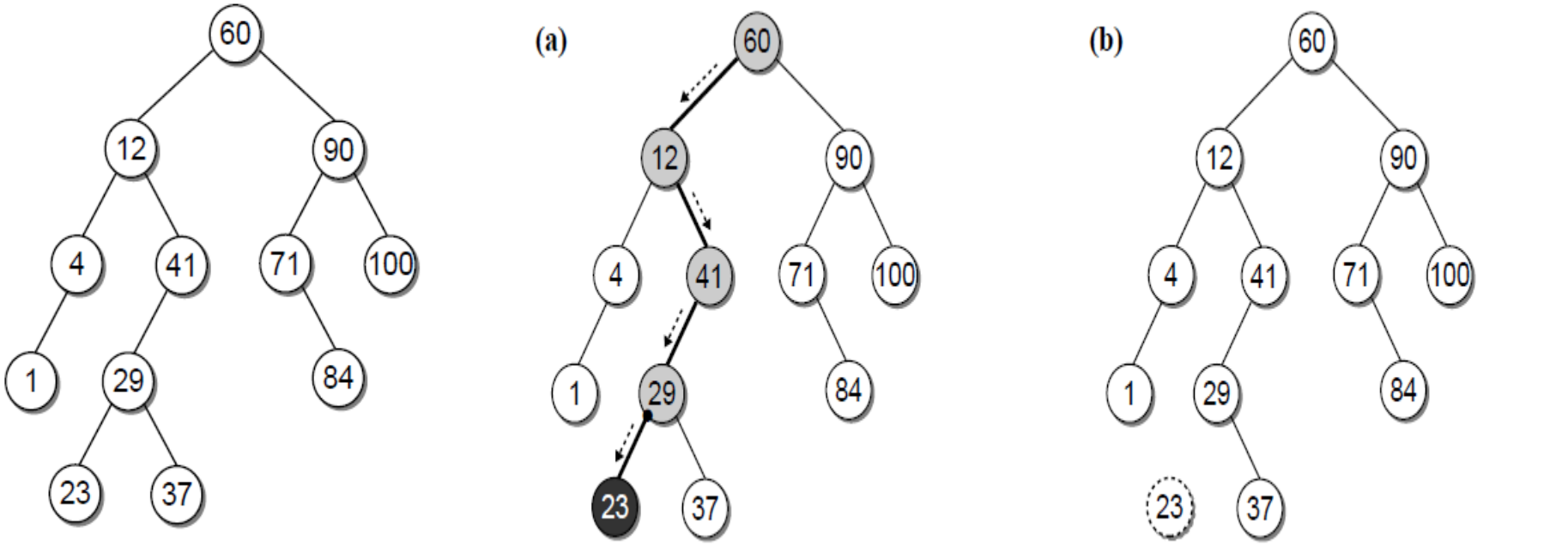
Deletion Operation in BST

- A deletion involves **searching for the node** that contains the target key and then **unlinking the node** to remove it from the tree.
- When a node is removed, the remaining nodes must preserve the search tree property.
- There are three cases to consider.
  1. The node is a leaf (no children).
  2. The node has a single child.
  3. The node has two children.

Removing a Leaf Node

- If the node to be deleted is the leaf node, then simply delete the node from the tree.

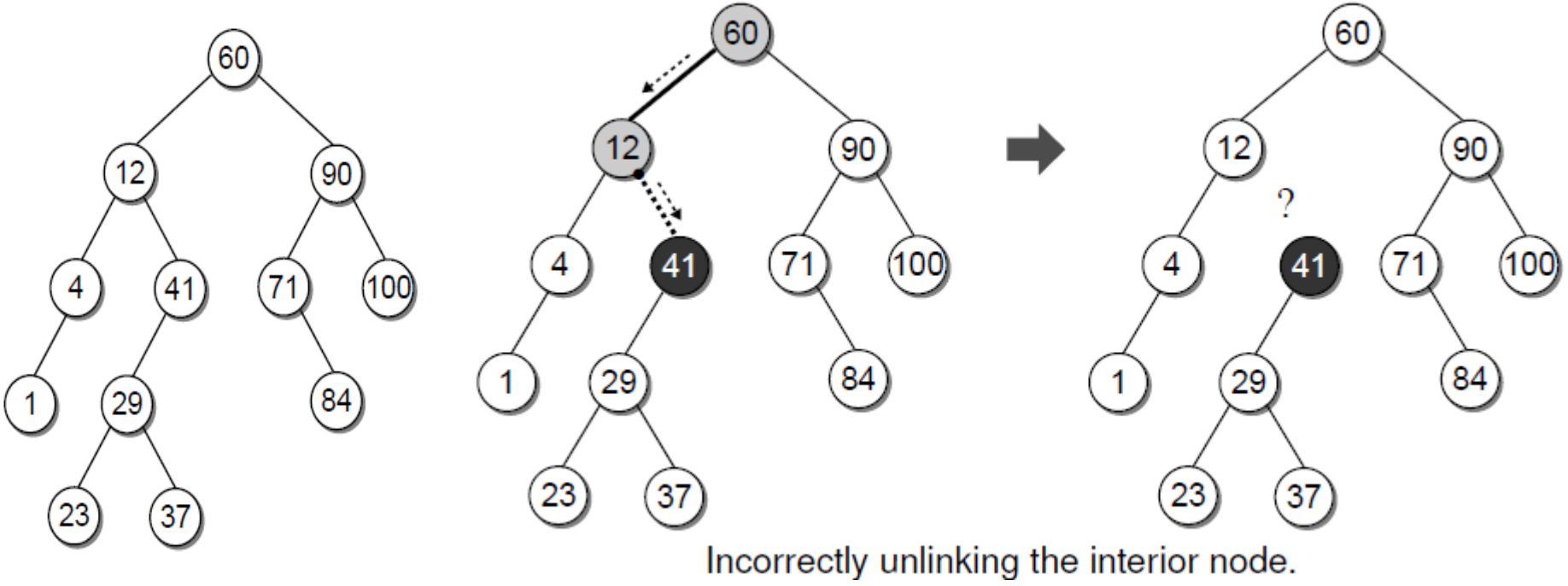
Suppose we want to delete key value 23



(a) finding the node and unlinking it from its parent; and (b) the tree after removing 23.

Removing an Interior Node with One Child

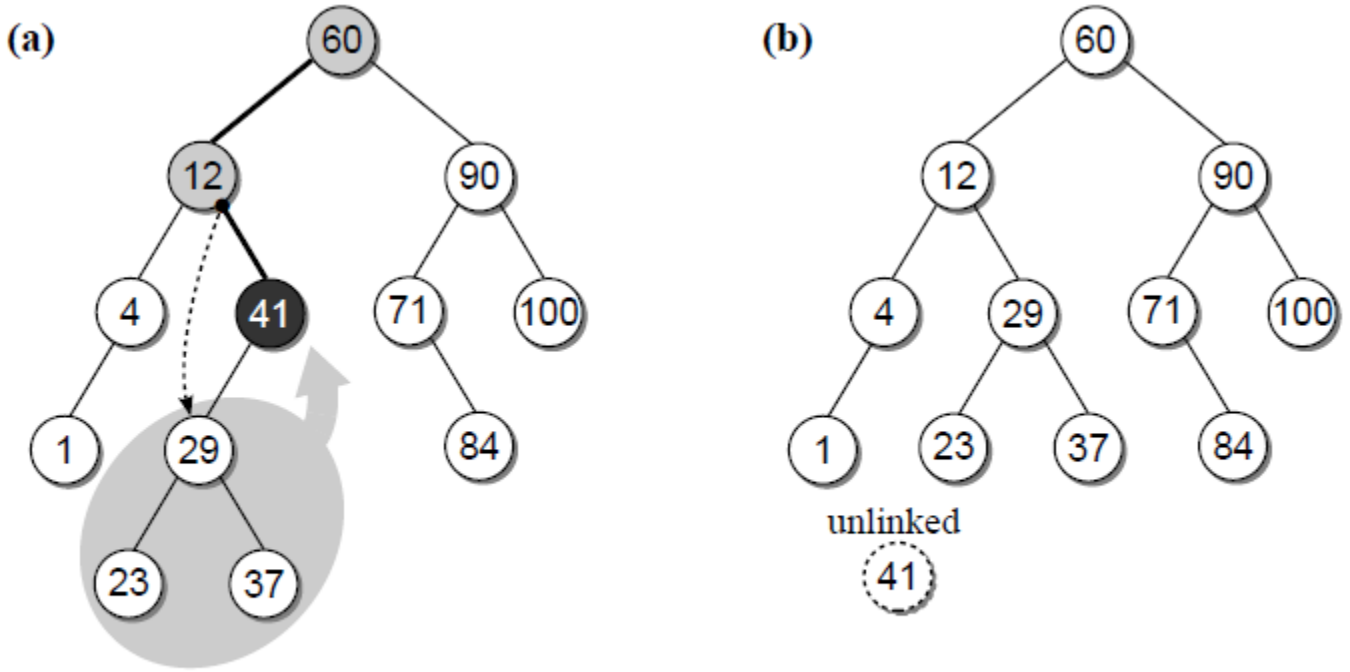
Suppose we want to delete key value 41



Incorrectly unlinking the interior node.

Removing an Interior Node with One Child

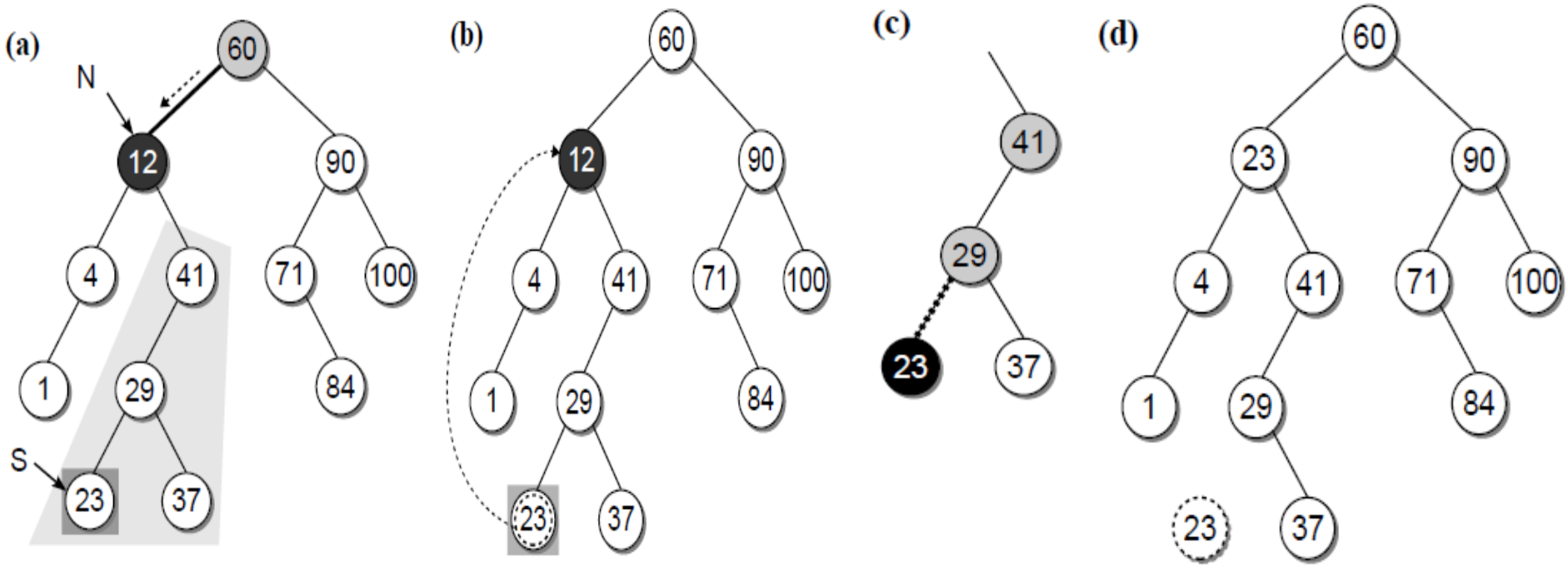
- If the node to be deleted lies has a single child node then,
  1. Replace that node with its child node.
  2. Remove the child node from its original position.



Removing an interior node (41) with one child: (a) redirecting the link from the node's parent to its child subtree; and (b) the tree after removing 41.

Removing an Interior Node with Two Children

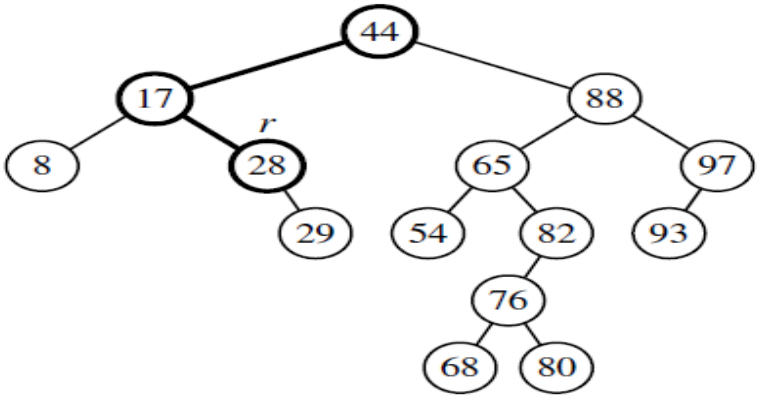
- If the node to be deleted has two children then,
  1. Get the inorder successor (S) of that node (N).
  2. Replace the node (N) with the inorder successor (S).
  3. Remove the inorder successor (S) from its original position.

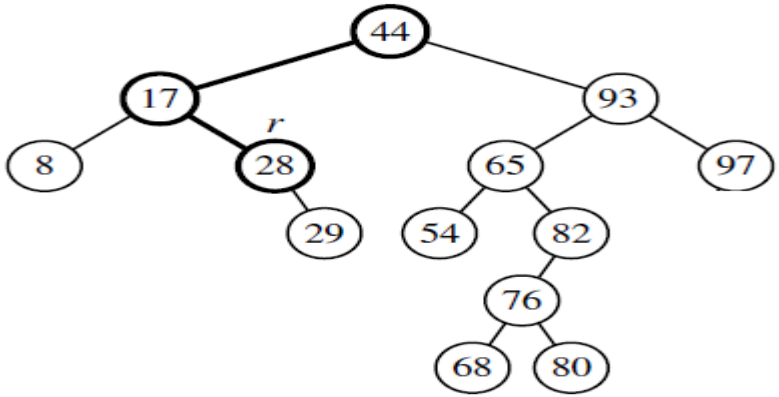


The steps in removing a key from a binary search tree: (a) find the node, N, and its successor, S; (b) copy the successor key from node N to S; (c) remove the successor key from the right subtree of N; and (d) the tree after removing 12.

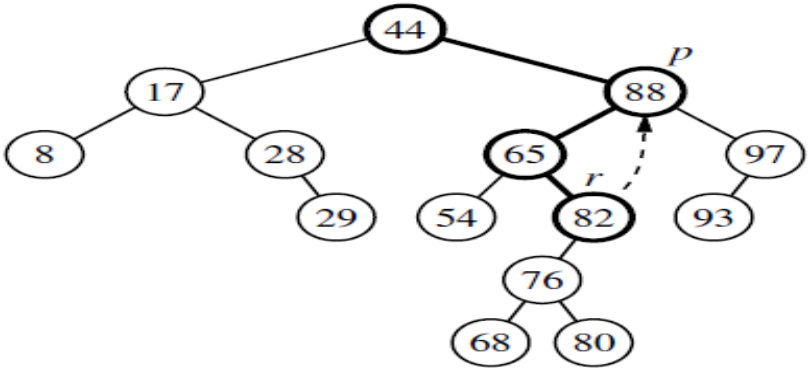
Removing an Interior Node with Two Children

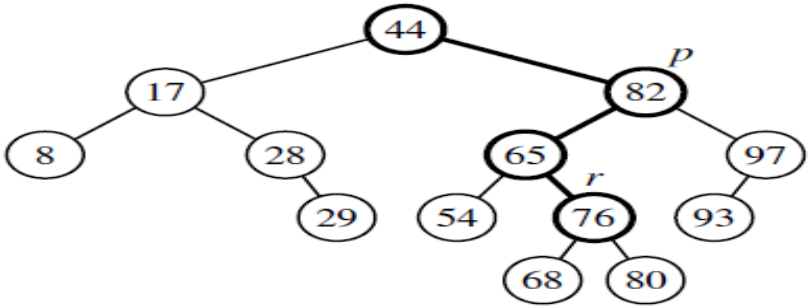
Suppose we want to delete key value 88





Deletion using inorder predecessor



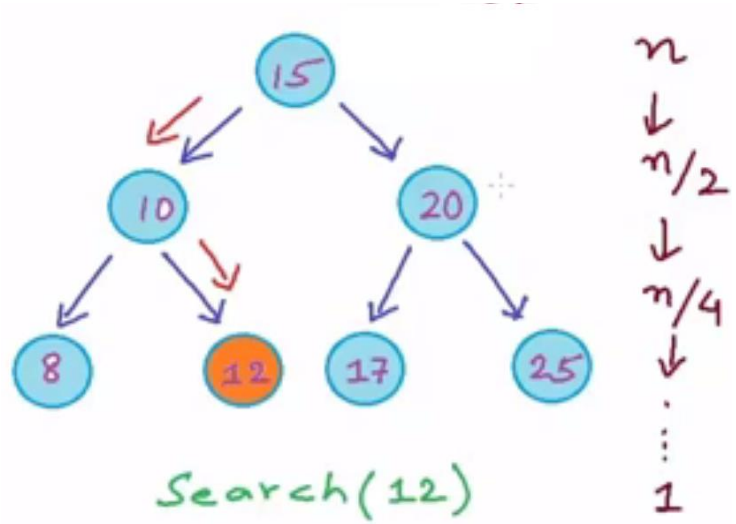


(a) (b)

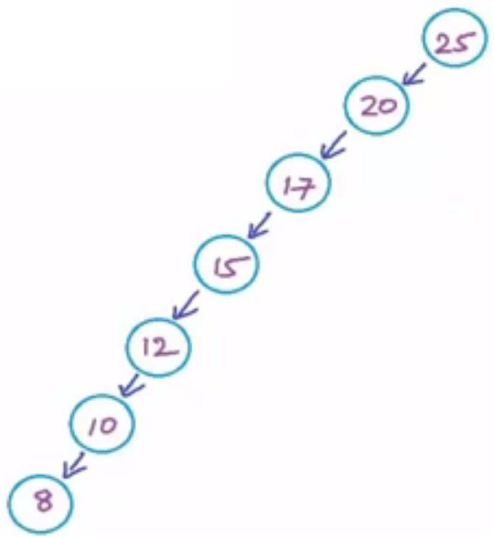
Deletion from the binary search tree, where the item to delete (with key 88) is stored at a position p with two children, and replaced by its inorder predecessor r: (a) before the deletion; (b) after the deletion.

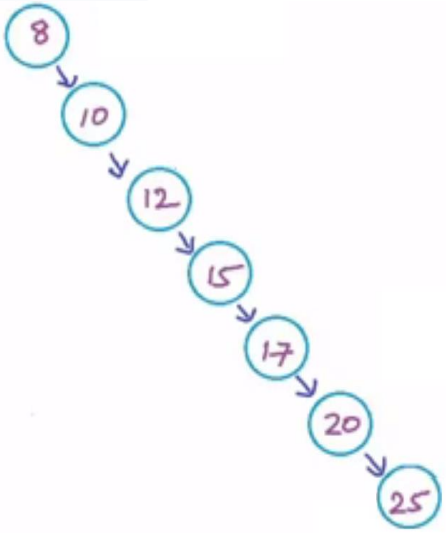
BST - Time Complexity

Operation	Best Case Complexity	Average Case Complexity	Worst Case Complexity
Search	O(log n)	O(log n)	O(n)
Insertion	O(log n)	O(log n)	O(n)
Deletion	O(log n)	O(log n)	O(n)



Best Case



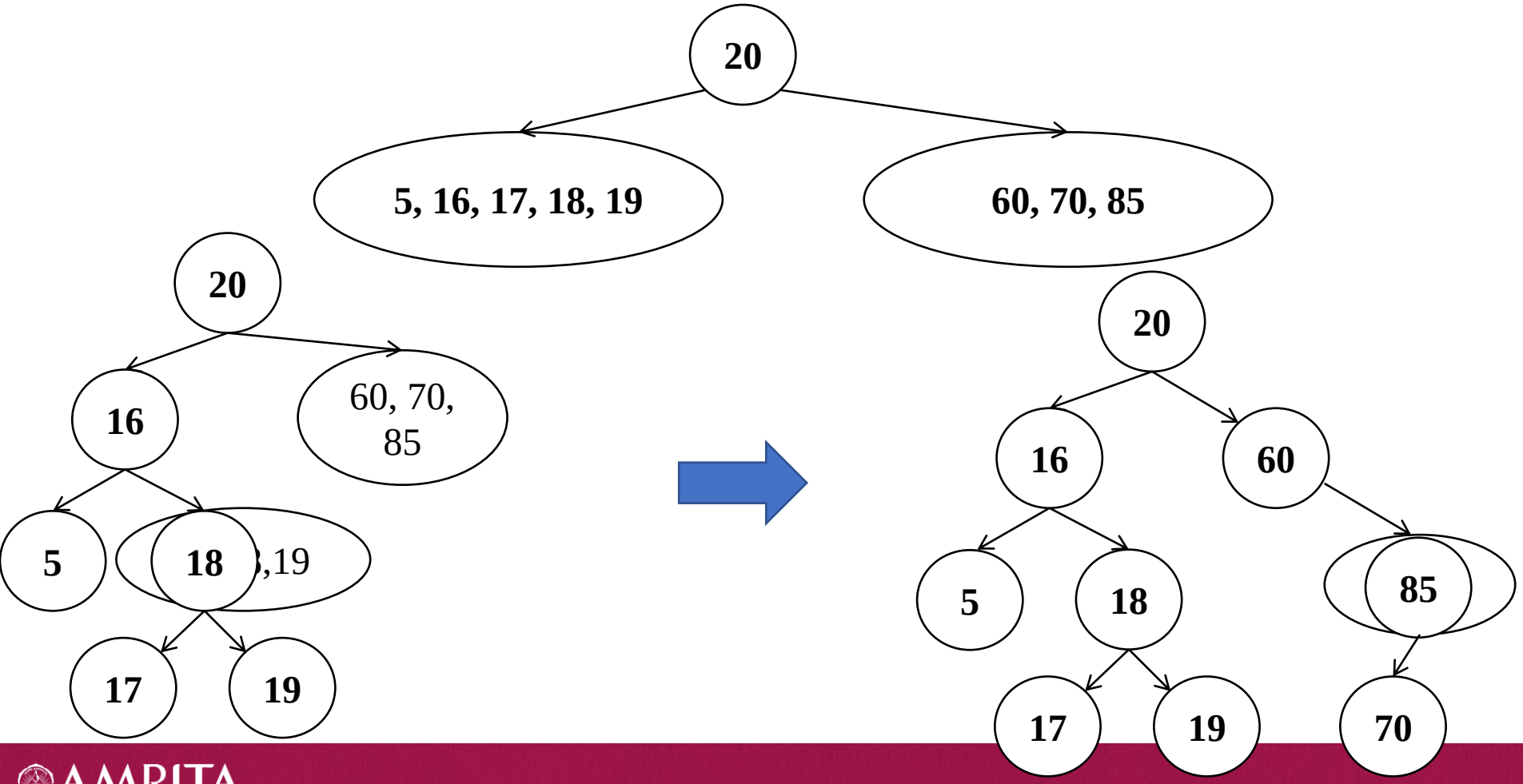


Worst Cases

Constricting a BST: Only preorder is given

Preorder: 20, 16, 5, 18, 17, 19, 60, 85, 70 (root, left, right)

Inorder: 5, 16, 17, 18, 19, **20**, 60, 70, 85 (left, root, right)



Constricting a BST: Only postorder is given

Preorder: 5, 17, 19, 18, 16, 70, 85, 60, 20 (left, right, root)

Postorder: 5, 16, 17, 18, 19, **20**, 60, 70, 85 (left, root, right)