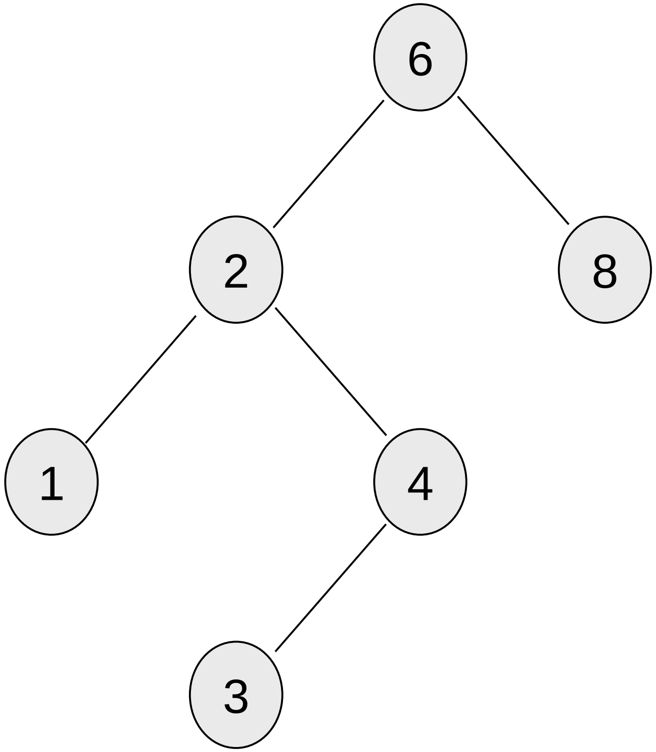
Binary Trees – Issues in Construction

- How can we insert a node in to a binary tree?
  - Need to specify the location as a left or right child of an existing node in the tree
  - What needs to be done if a node is already present at that location?
  - The tree constructed can be of height n -1 (n is number of nodes in the tree
  - The operations of insertion, search and deletion can be of complexity O(n).
- ***Binary search tree*** is an alternative to make the searching convenient and also with average time complexity of O(log n).

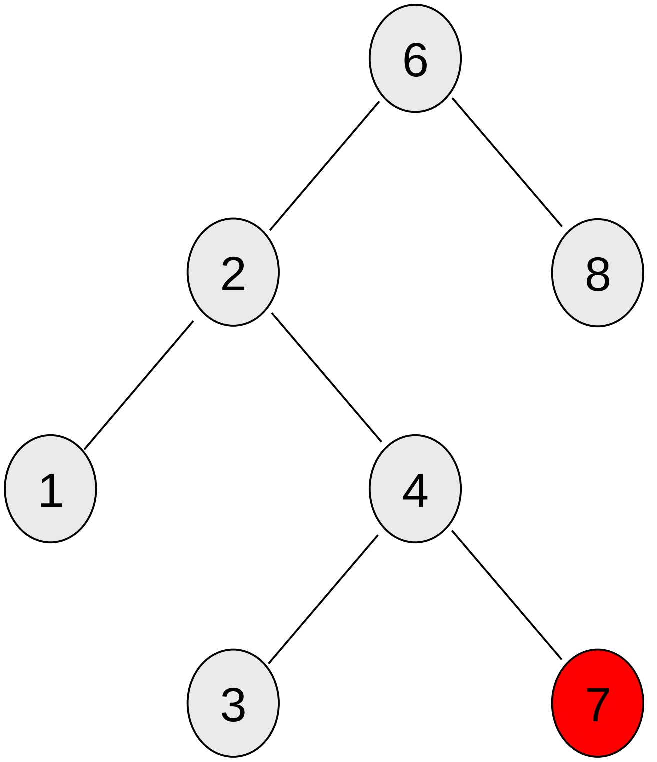
Binary Search Trees

- An important application of binary trees is their use in searching.
- ***Binary search tree*** is a binary tree in which every node X contains a data value that satisfies the following:
  a) all data values in its left subtree are smaller than the data value in X
  b) the data value in X is smaller than all the values in its right subtree.
  c) the left and right subtrees are also binary search tees.

Example

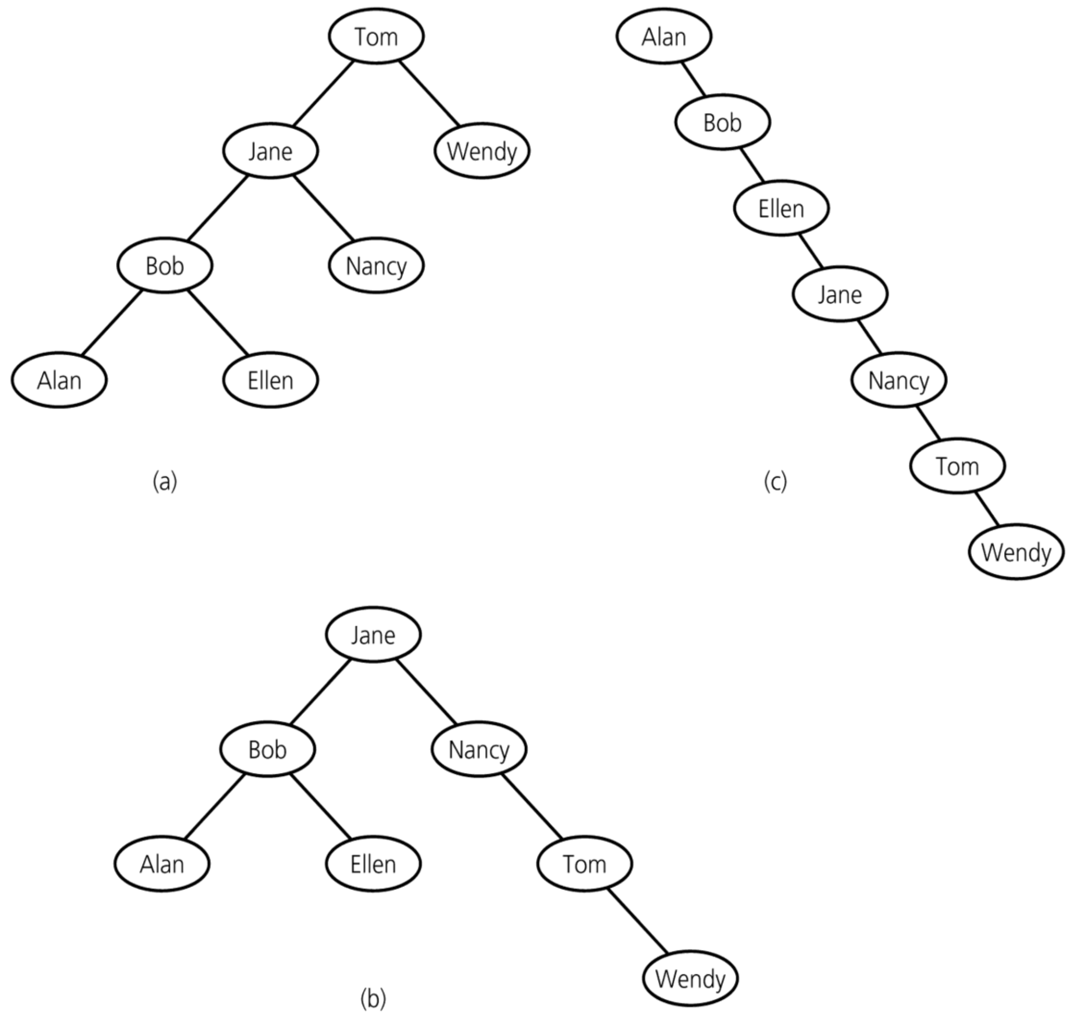


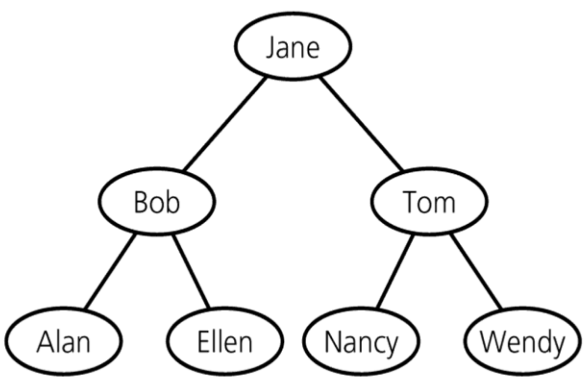
A *binary search tree*



Not a *binary search tree,* but a *binary tree*

Binary Search Trees – containing same data





Operations on BSTs

- Most of the operations on binary trees are O(log$N$).
  - This is the main motivation for using binary trees rather than using ordinary lists to store items.
- Most of the operations can be implemented using recursion.
  - we generally do not need to worry about running out of stack space, since the average depth of binary search trees is O(log$N$).

Insert operation

Algorithm for inserting X into tree T:

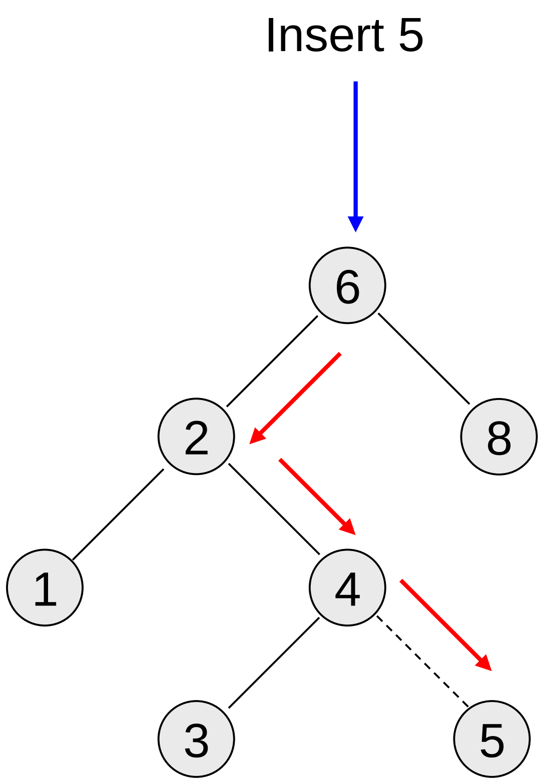
- Proceed down the tree as you would with a find operation.
- if X is found

  do nothing, (or "update" something)

  else

  insert X at the last spot on the path traversed.

Example



- What about duplicates?

Deletion operation

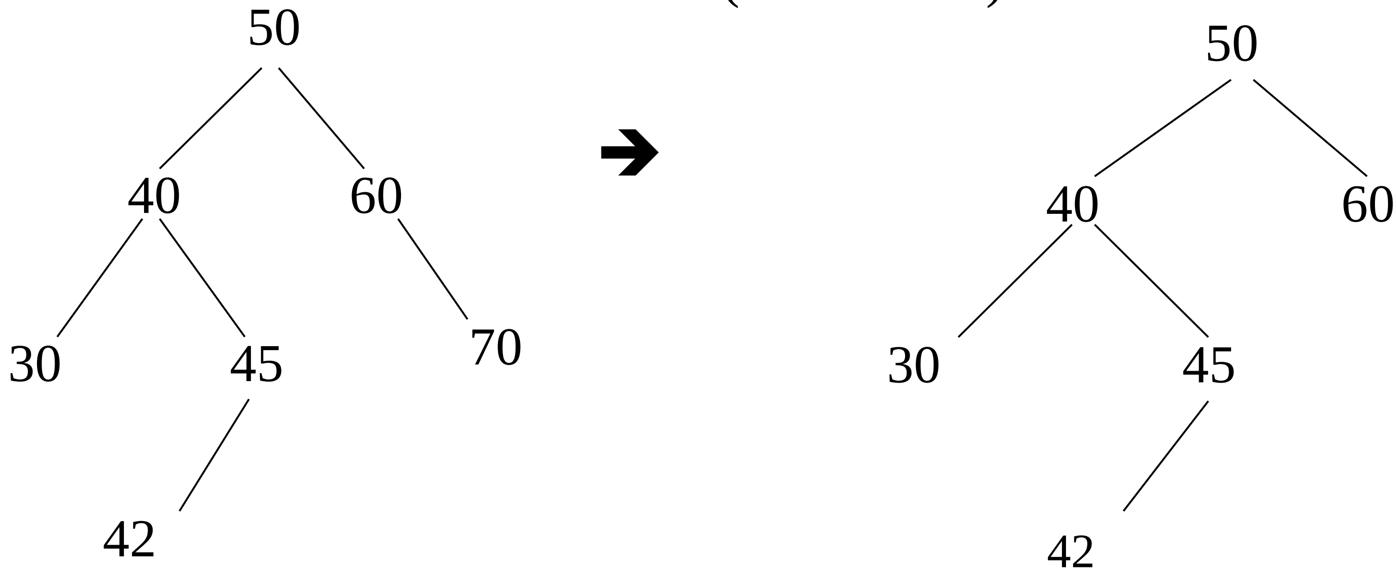
There are three cases to consider:

1. Deleting a leaf node
   - Replace the link to the deleted node by NULL.
2. Deleting a node with one child:
   - The node can be deleted after its parent adjusts a link to bypass the node.
3. Deleting a node with two children:
   - The deleted value must be replaced by an existing value that is either one of the following:
     - The largest value in the deleted node’s left subtree
     - The smallest value in the deleted node’s right subtree.

Deletion – Case1: A Leaf Node

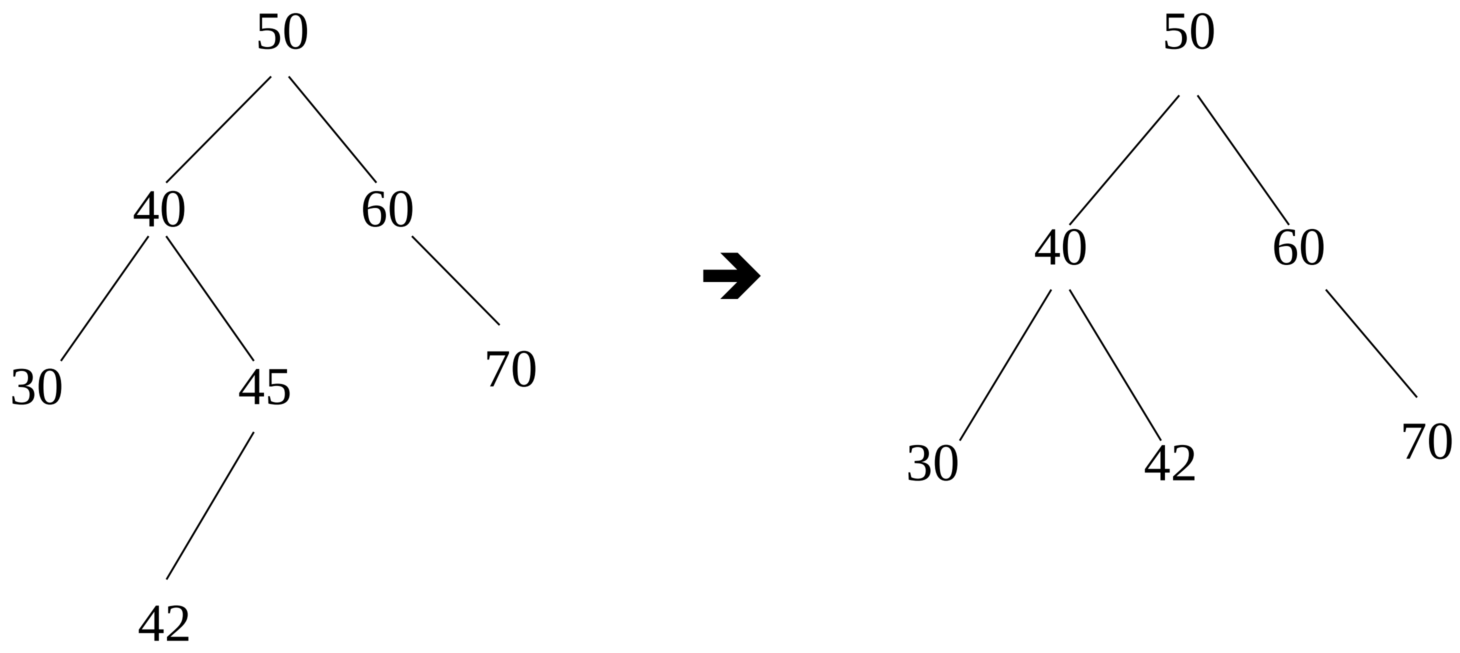
To remove the leaf containing the item, we have to set the pointer in its parent to NULL.

Delete 70 (A leaf node)



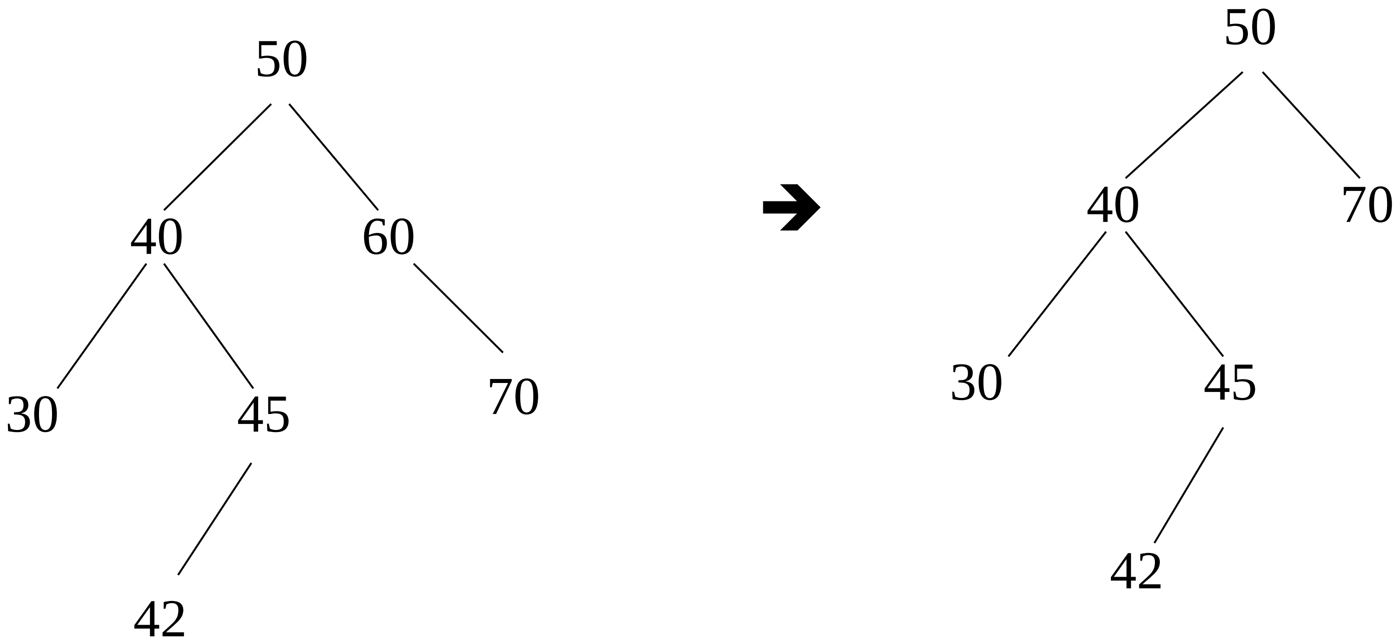
Deletion – Case2: A Node with only a left child

Delete 45 (A node with only a left child)



Deletion – Case2: A Node with only a right child

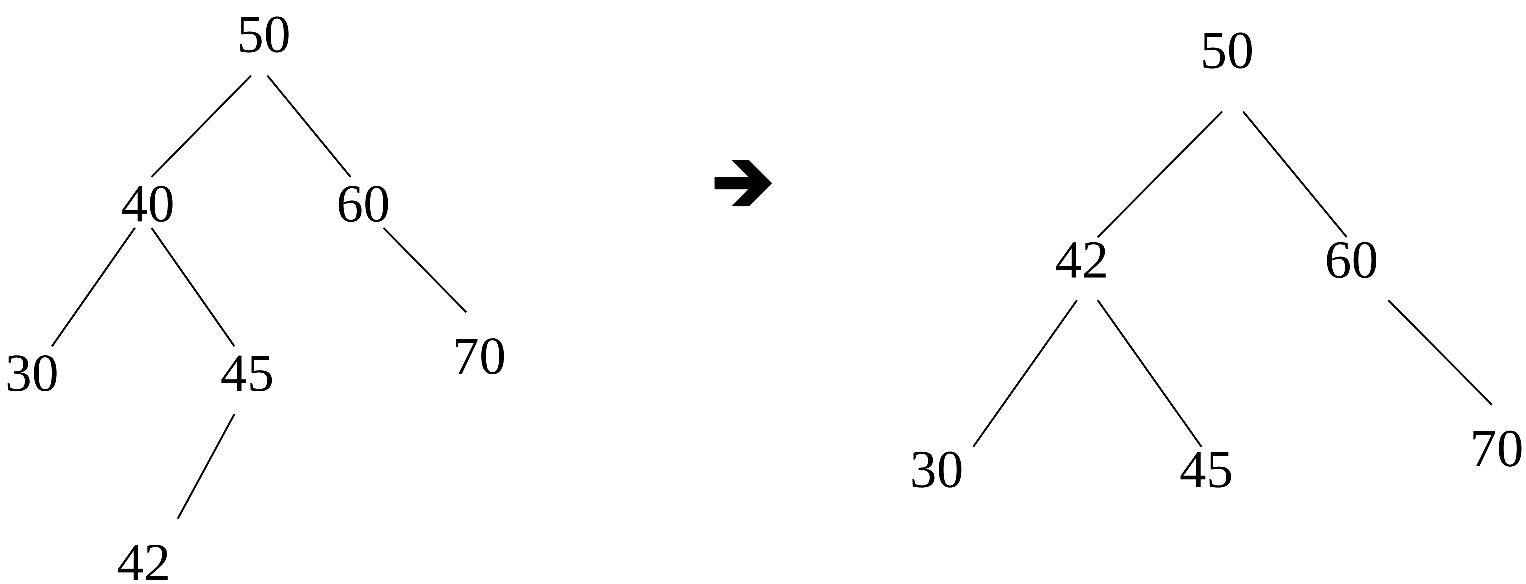
Delete 60 (A node with only a right child)



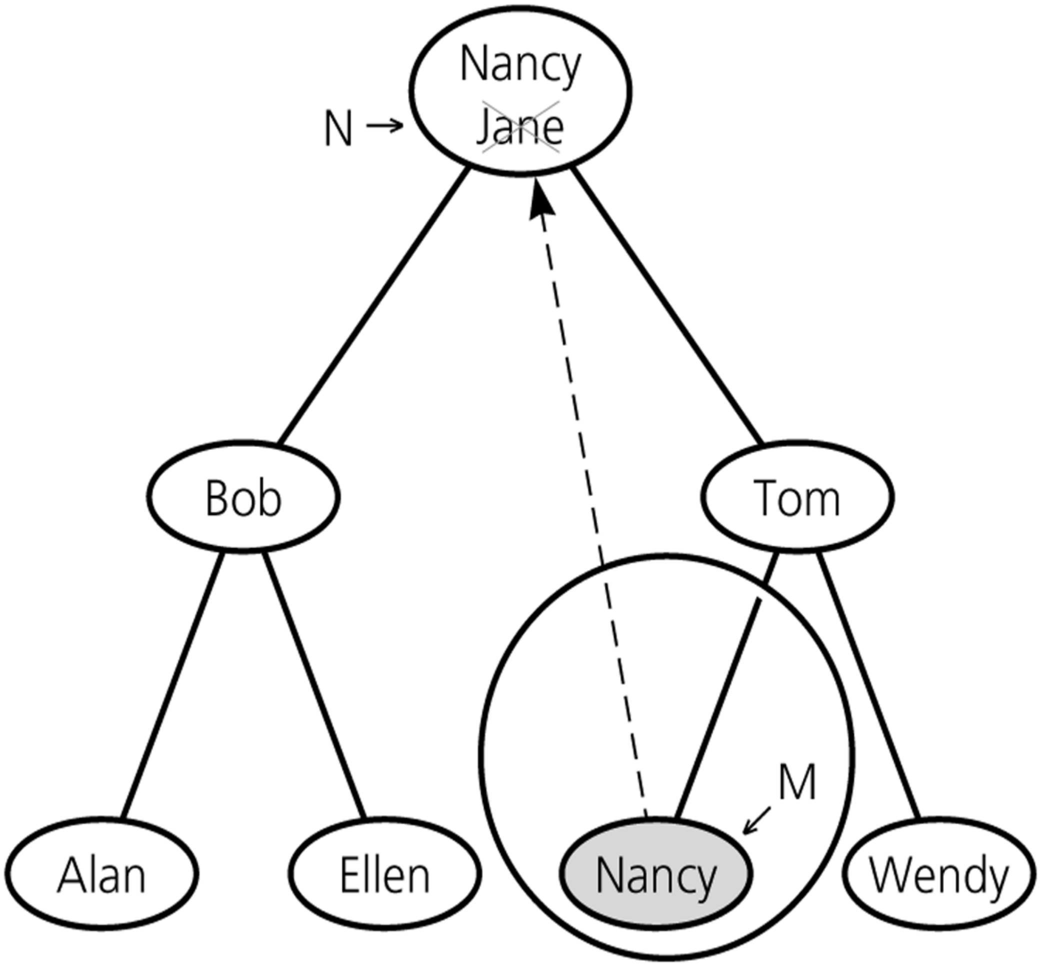
Deletion – Case3: A Node with two children

- Locate the inorder successor of the node.
- Copy the item in this node into the node which contains the item which will be deleted.
- Delete the node of the inorder successor.

Delete 40 (A node with two children)



Deletion – Case3: A Node with two children



Analysis of BST Operations

- The cost of an operation is proportional to the depth of the last accessed node.
- The cost is logarithmic for a well-balanced tree, but it could be as bad as linear for a degenerate tree.
- In the best case we have logarithmic access cost, and in the worst case we have linear access cost.

Order of Operations on BSTs

Operation	Average case	Worst case
Retrieval	O(log n)	O(n)
Insertion	O(log n)	O(n)
Deletion	O(log n)	O(n)
Traversal	O(n)	O(n)