Binary Search Tree

Dr. Manmath Narayan Sahoo
Dept. of CSE, NIT Rourkela

Binary Search Tree (BST)

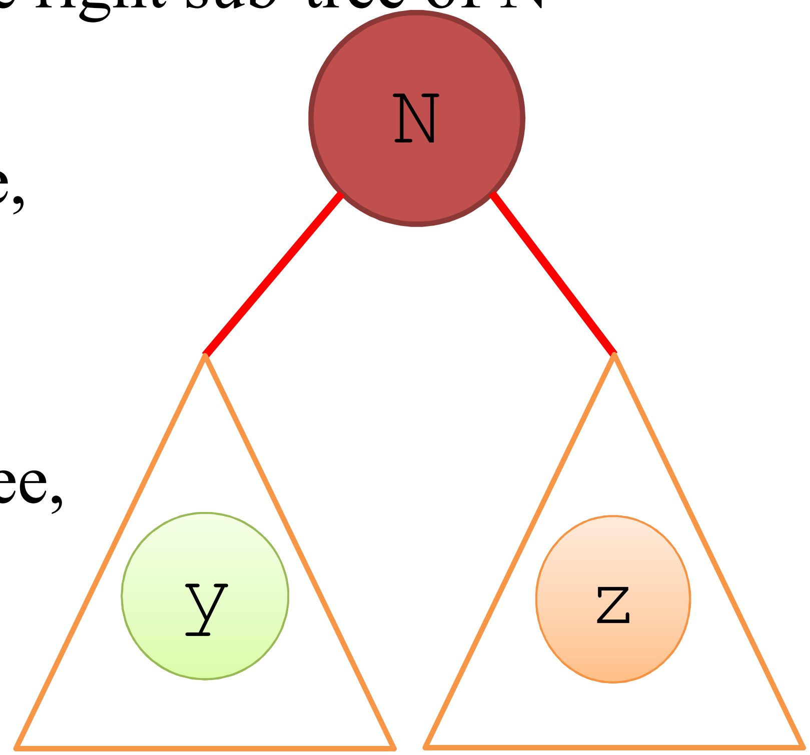
- A binary tree T is a binary search tree if the value at each node N is greater than every value in the left sub-tree of N and is less than every value in the right sub-tree of N

- For any node `y` in the left subtree,

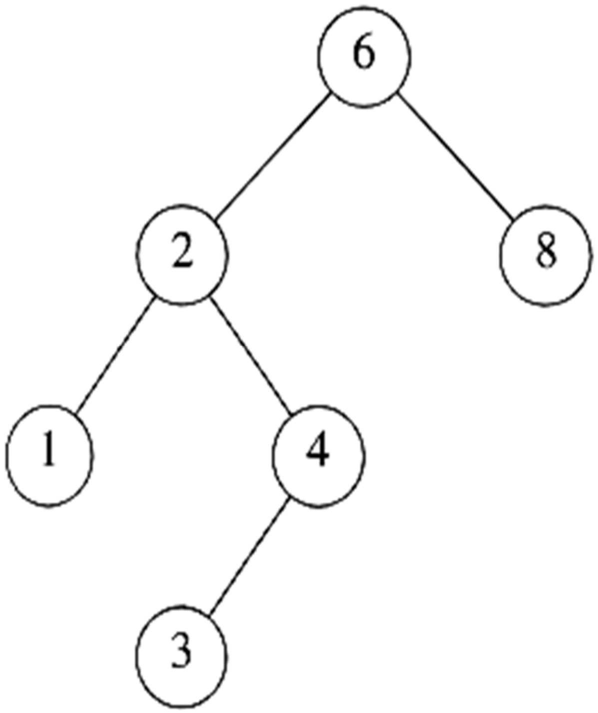
  `key(y) < key(N)`

- For any node `z` in the right subtree,

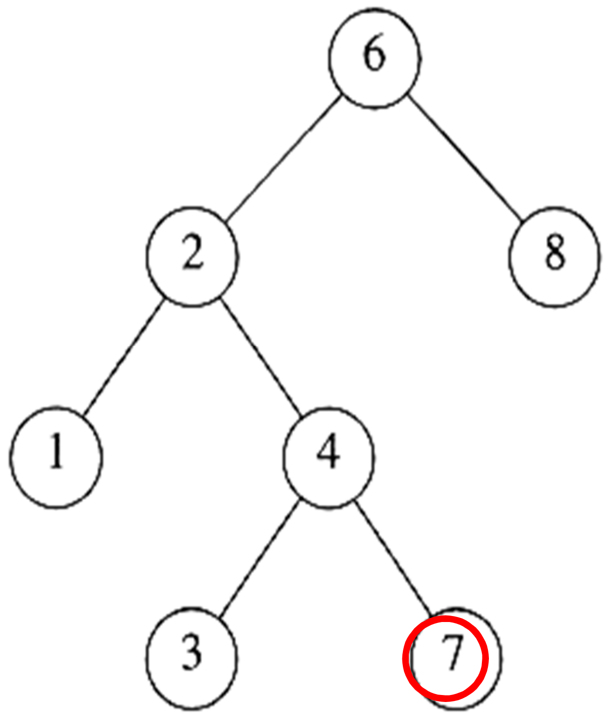
  `key(z) > key(N)`



Binary Search Trees



A binary search tree



Not a binary search tree

Searching and Inserting in BST

Algorithm to find the location of **ITEM** in the BST **T** or insert **ITEM** as a new node in its appropriate place in the tree

[a] Compare ITEM with the root node N of the tree

(i) If ITEM < N, proceed to the left child of N

(ii) If ITEM > N, proceed to the right child of N

Searching and Inserting in BST

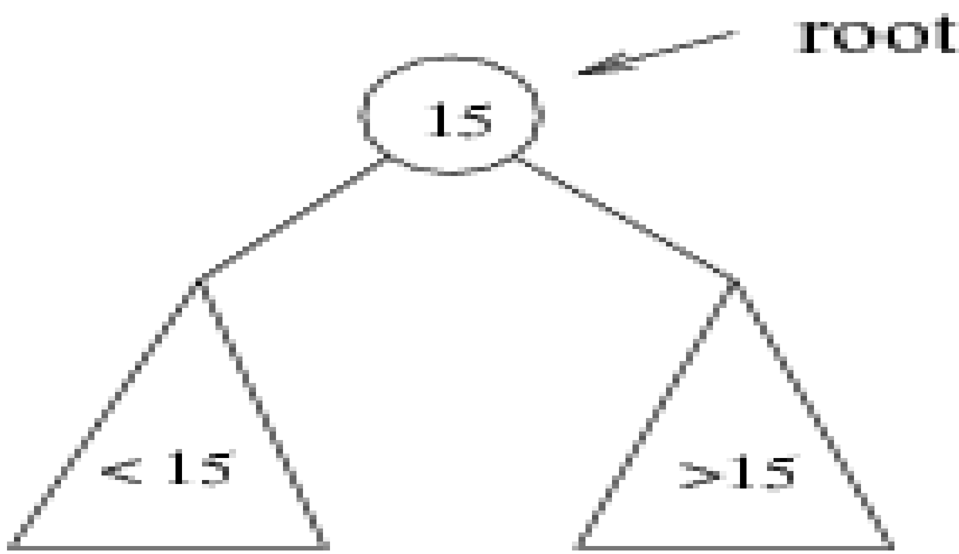
[b] Repeat Step (a) until one of the following occurs

(i) We meet a node N such that ITEM = N. In this case search is successful

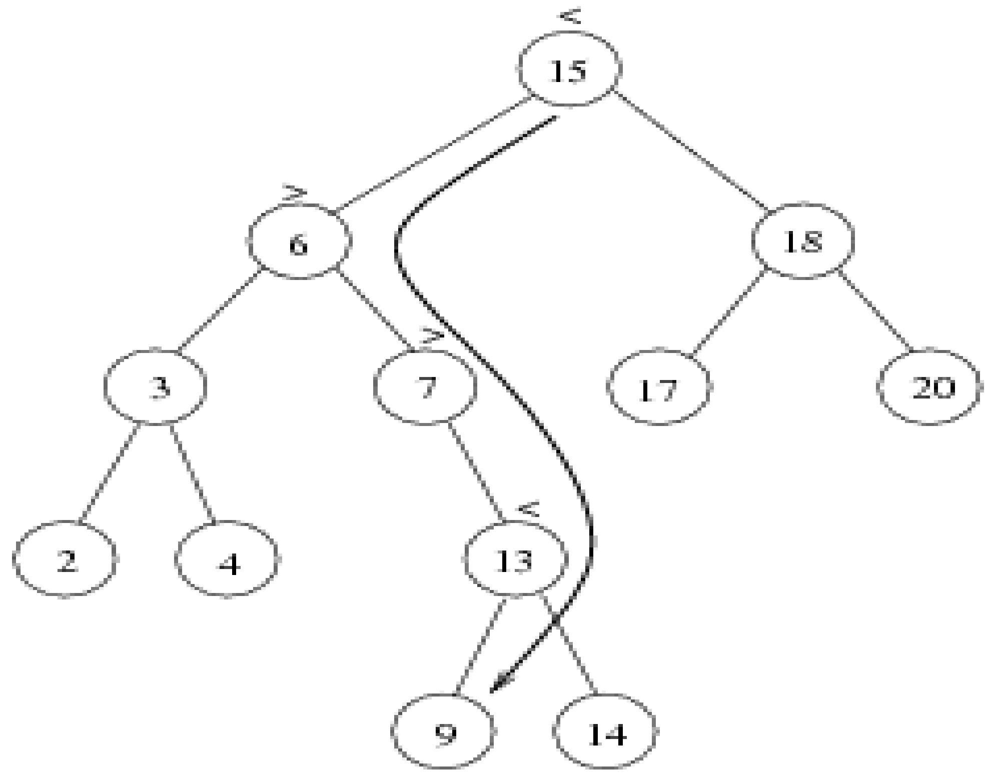
(ii) We meet an empty sub-tree, which indicates that search is unsuccessful and we insert ITEM in place of empty subtree

Searching BST

- If we are searching for 15, then we are done.
- If we are searching for a key < 15, then we should search in the left subtree.
- If we are searching for a key > 15, then we should search in the right subtree.



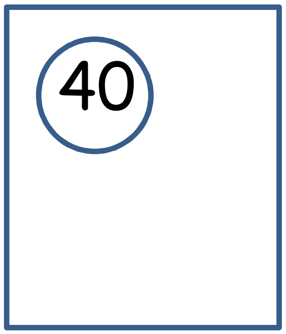
Example: Search for 9 ...



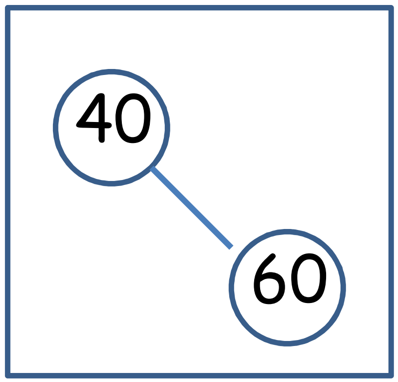
Search for 9:

1. compare 9:15(the root), go to left subtree;
2. compare 9:6, go to right subtree;
3. compare 9:7, go to right subtree;
4. compare 9:13, go to left subtree;
5. compare 9:9, found it!

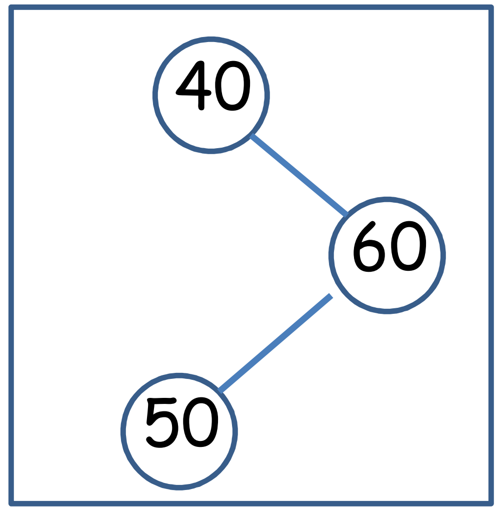
Insert 40, 60, 50, 33, 55, 11 into an empty BST



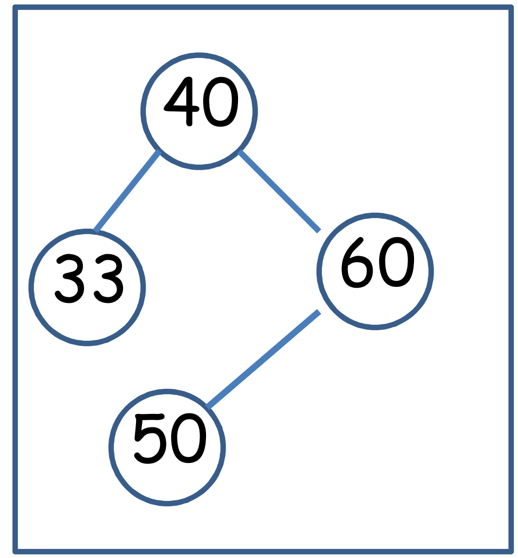
1. ITEM = 40



2. ITEM = 60

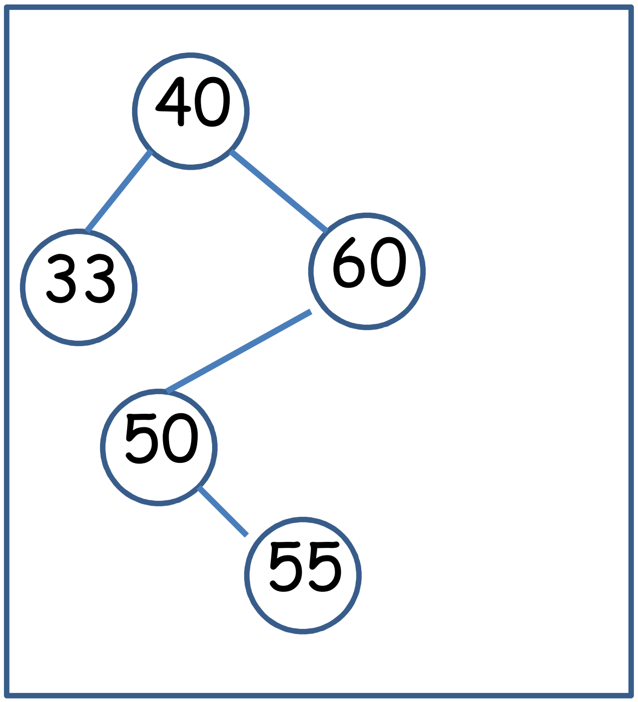


3. ITEM = 50

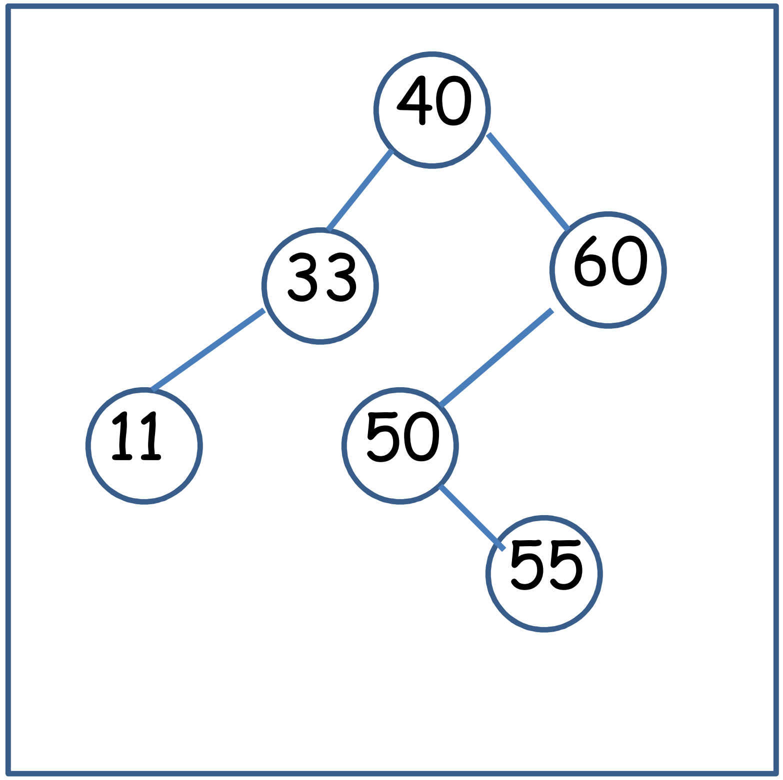


4. ITEM = 33

Insert 40, 60, 50, 33, 55, 11 into an empty BST



5. ITEM = 55



6. ITEM = 11

Locating an ITEM

A binary search tree T is in memory and an ITEM of information is given. This procedure finds the location **LOC** of **ITEM** in T and also the location of the parent **PAR** of ITEM.

FIND(INFO,LEFT,RIGHT,ROOT,ITEM,LOC,PAR)

[1] [Tree empty ?]

If ROOT == NULL, then

Set LOC = NULL, PAR = NULL, Exit

[2] [ITEM at root ?]

If ROOT→INFO == ITEM, then

Set LOC = ROOT, PAR = NULL, Exit

[3] [Initialize pointer PTR and SAVE]

If ITEM < ROOT→INFO then

Set PTR = ROOT→LEFT, SAVE = ROOT

Else

Set PTR = ROOT→RIGHT, SAVE =ROOT

[4] Repeat Step 5 and 6 while PTR $\neq$ NULL

[5] [ITEM Found ?]

If ITEM == PTR→INFO, then

Set LOC = PTR, PAR = SAVE, Exit

[6] If ITEM < PTR→INFO, then

Set SAVE = PTR, PTR = PTR→LEFT

Else

Set SAVE = PTR, PTR = PTR→RIGHT

[7] [Search Unsuccessful]

Set LOC = NULL, PAR = SAVE

[8] Exit

Locating an ITEM

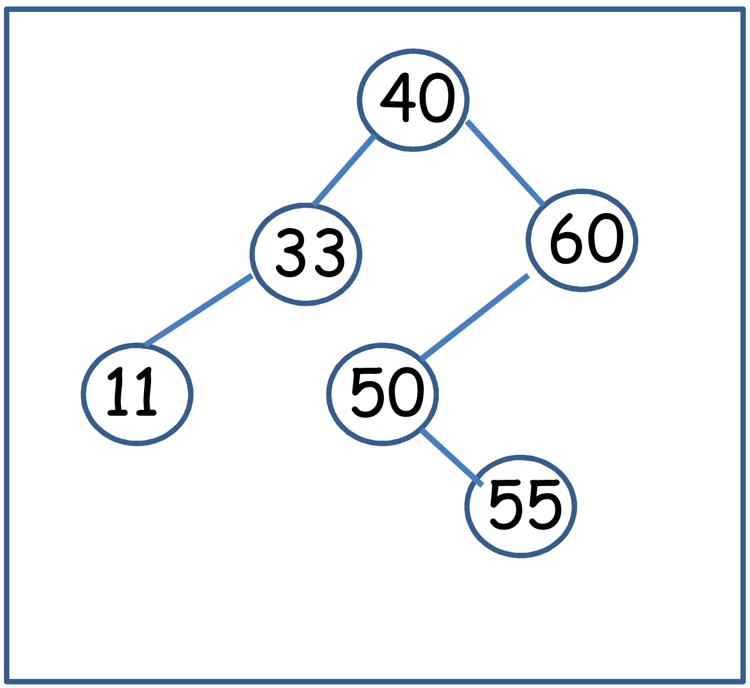
Four outcomes of FIND:

[1] LOC == NULL and PAR == NULL, tree is empty

[2] LOC ≠ NULL and PAR == NULL, ITEM is the root of T

[3] LOC == NULL and PAR ≠ NULL, ITEM is not in T and can be added to T as child of the node N with location PAR.

[4] LOC ≠ NULL and PAR ≠ NULL, ITEM is present but not at the root of T.



Insertion to BST

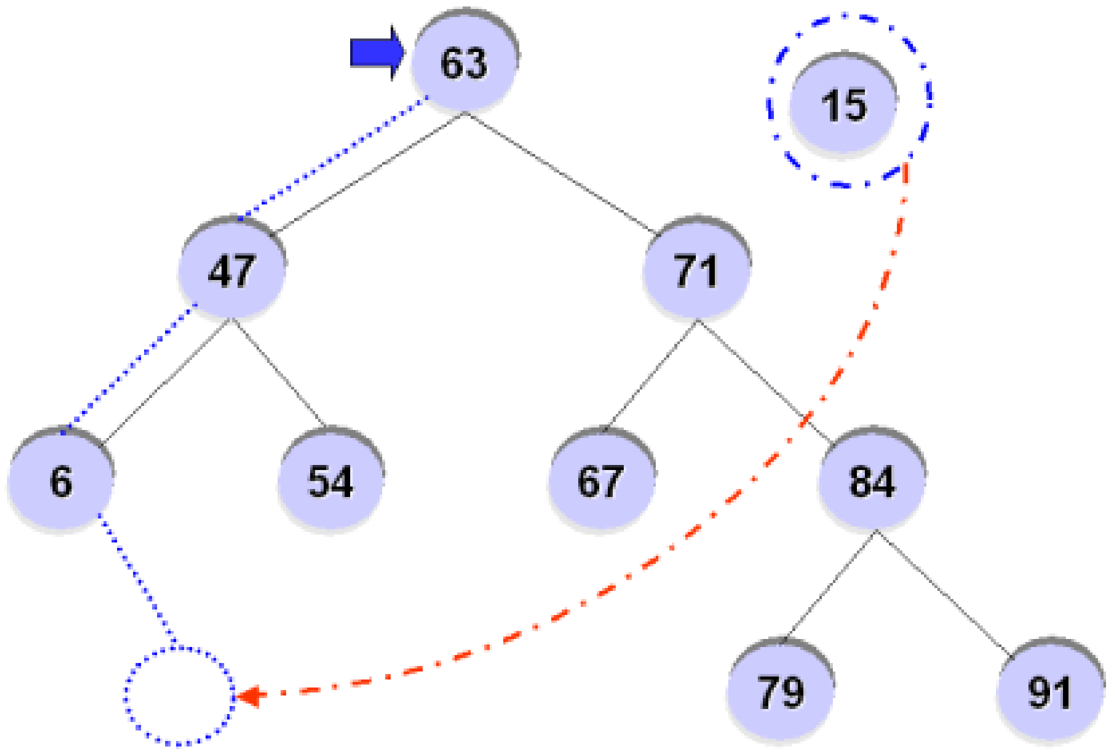
A binary search Tree T is in memory and an ITEM of information is given. Algorithm to add ITEM as a new node in T, if doesn’t exist.

Insertion Algorithm

[1] Call FIND(INFO, LEFT,RIGHT,ROOT,ITEM,LOC,PAR)
[2] If LOC ≠ NULL, then Exit //ITEM exists, no insertion
[3] [Copy the ITEM into the node NEW]
    (a) Create a node NEW
    (b) NEW→INFO = ITEM
    (c) Set LOC = NEW,
        NEW→LEFT = NULL,
        NEW→RIGHT = NULL
[4] [Add NEW node to tree]
    If PAR = NULL
        Set ROOT = NEW
    Else
        If ITEM < PAR→INFO, then
            Set PAR→LEFT = NEW
        Else
            Set PAR→RIGHT = NEW
[5] Exit

Binary Search Tree

For example, inserting '15' into the BST?



Deletion from BST

T is a binary search tree. Delete an ITEM from the tree T

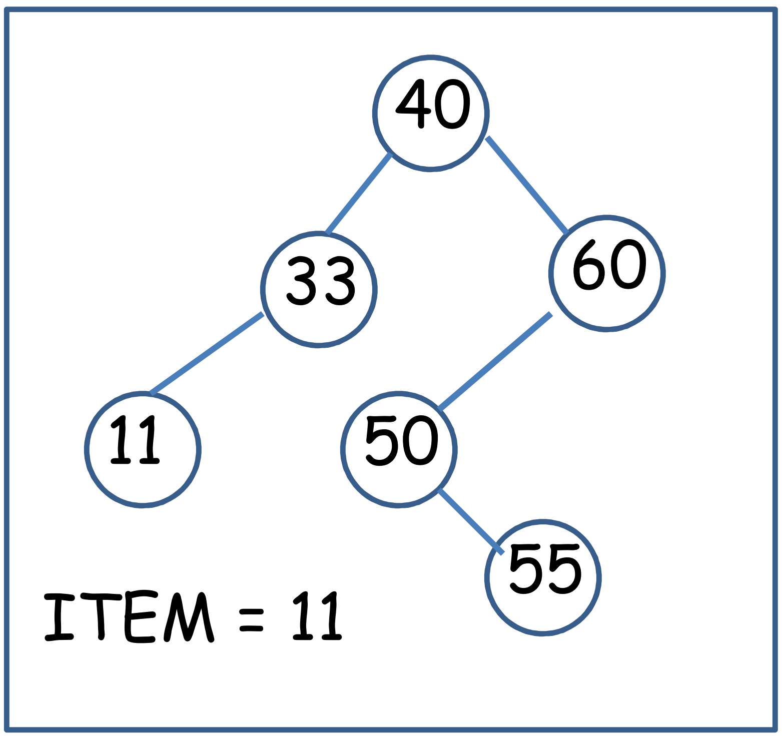
Deleting a node **N** from tree depends primarily on the number of children of node **N**

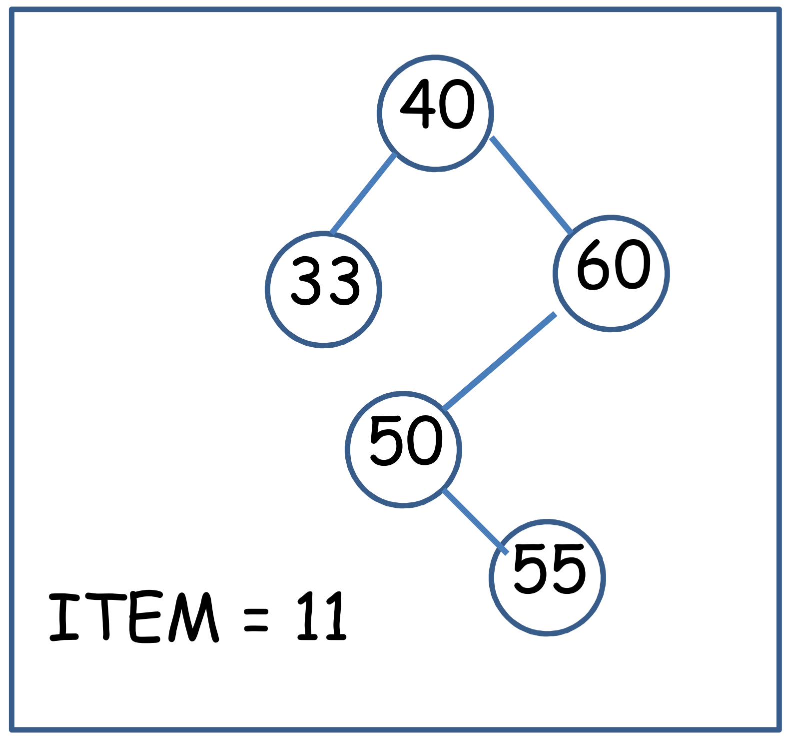
Deletion

There are three cases in deletion

Case 1. N has no children.

N is deleted from the T by replacing the location of N in the parent node of N by null pointer

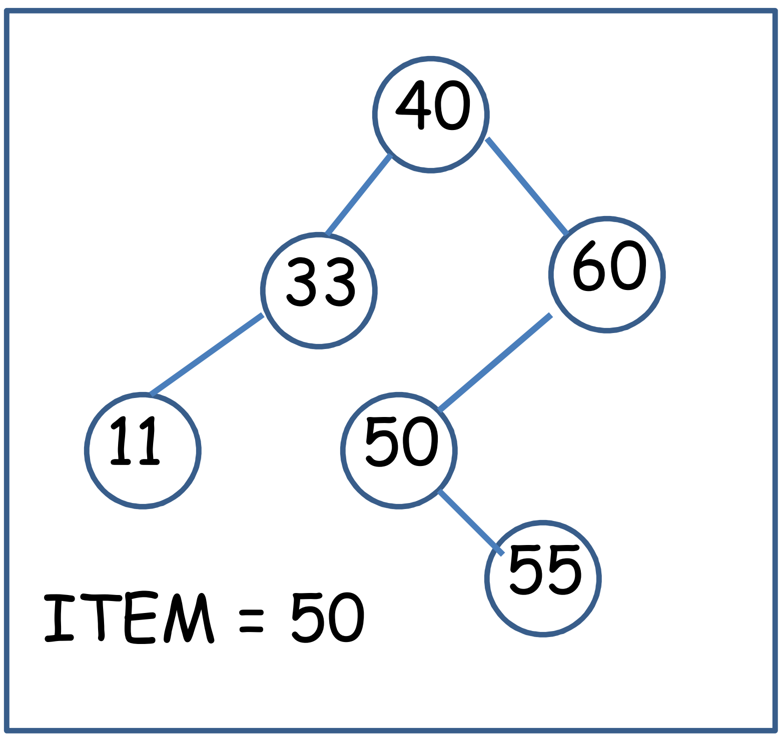


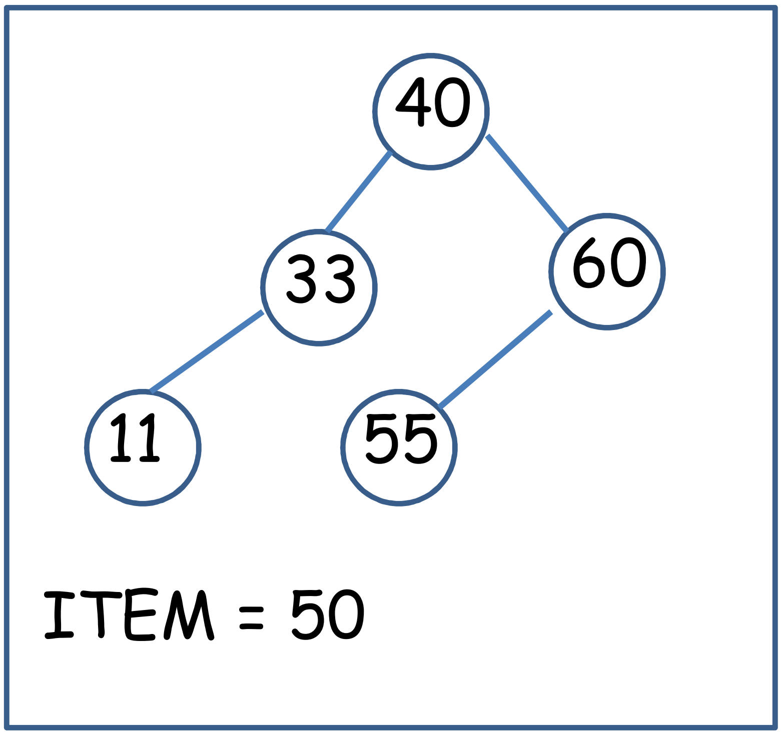


Deletion

Case 2. N has exactly one child.

N is deleted from the T by simply replacing the location of N in the parent node of N by the location of the only child of N





Deletion

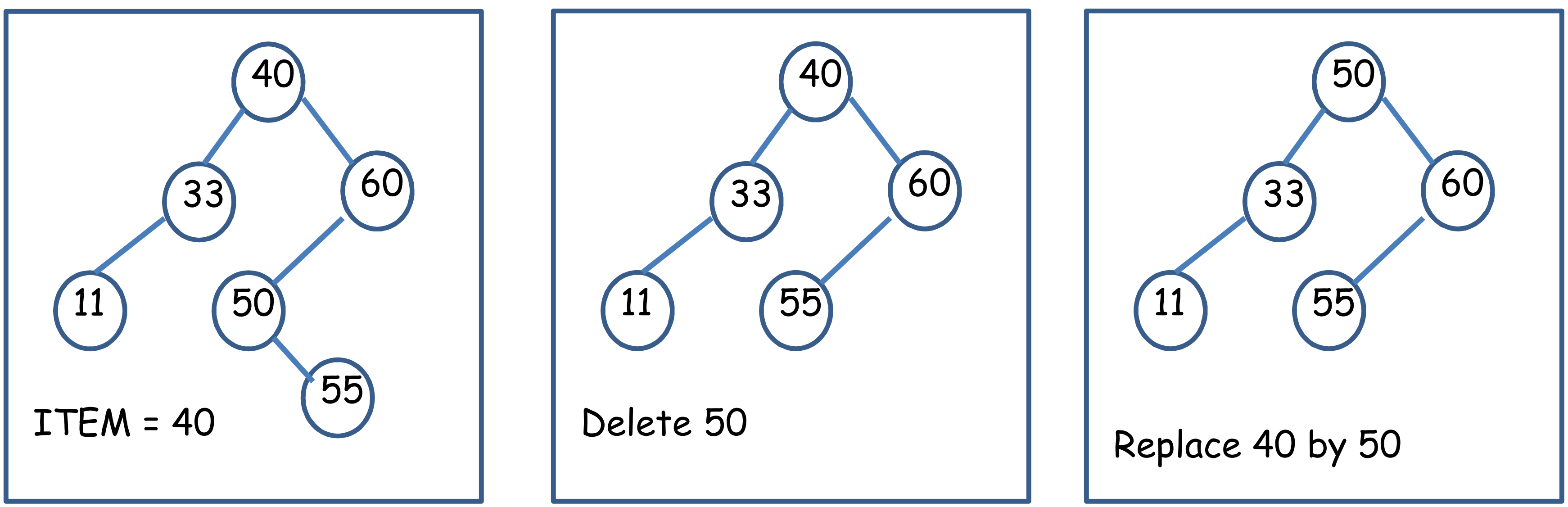
Case 3. N has two children.

Let S(N) denote the in-order successor of N.

Then N is deleted from the T by

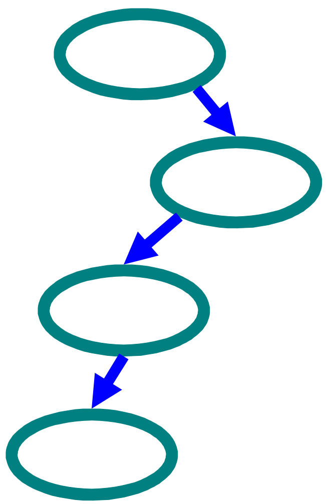
deleting S(N) from T and then

replacing node N in T by the node S(N)

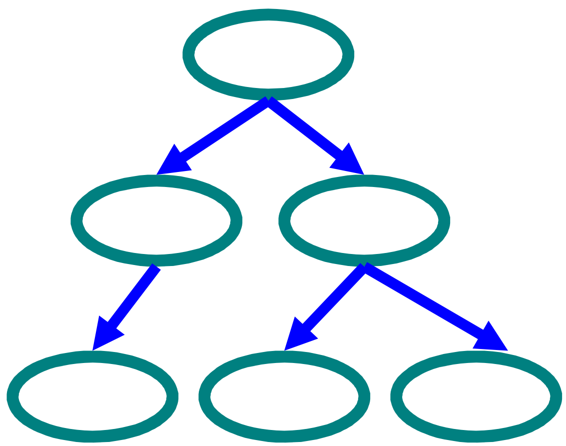


Types of Binary Trees

- Degenerate – only one child
- Balanced – mostly two children



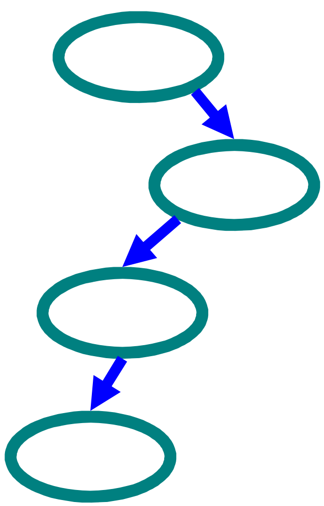
Degenerate binary tree



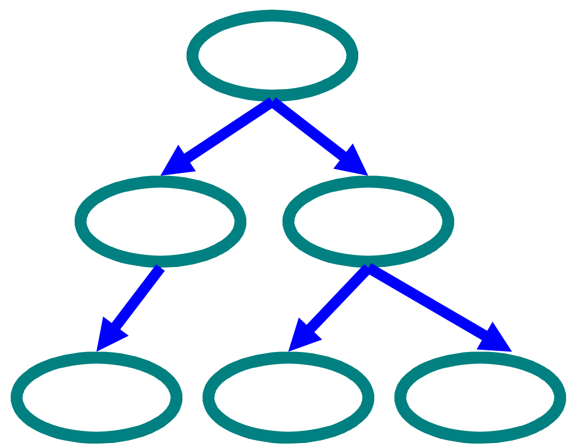
Balanced binary tree

Binary Trees Properties

- Degenerate
  - Height = O(n) for n nodes
  - Similar to linear list
- Balanced
  - Height = O(log(n)) for n nodes
  - Useful for searches



Degenerate binary tree



Balanced binary tree