Data Structure and Algorithm (CS-102)

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

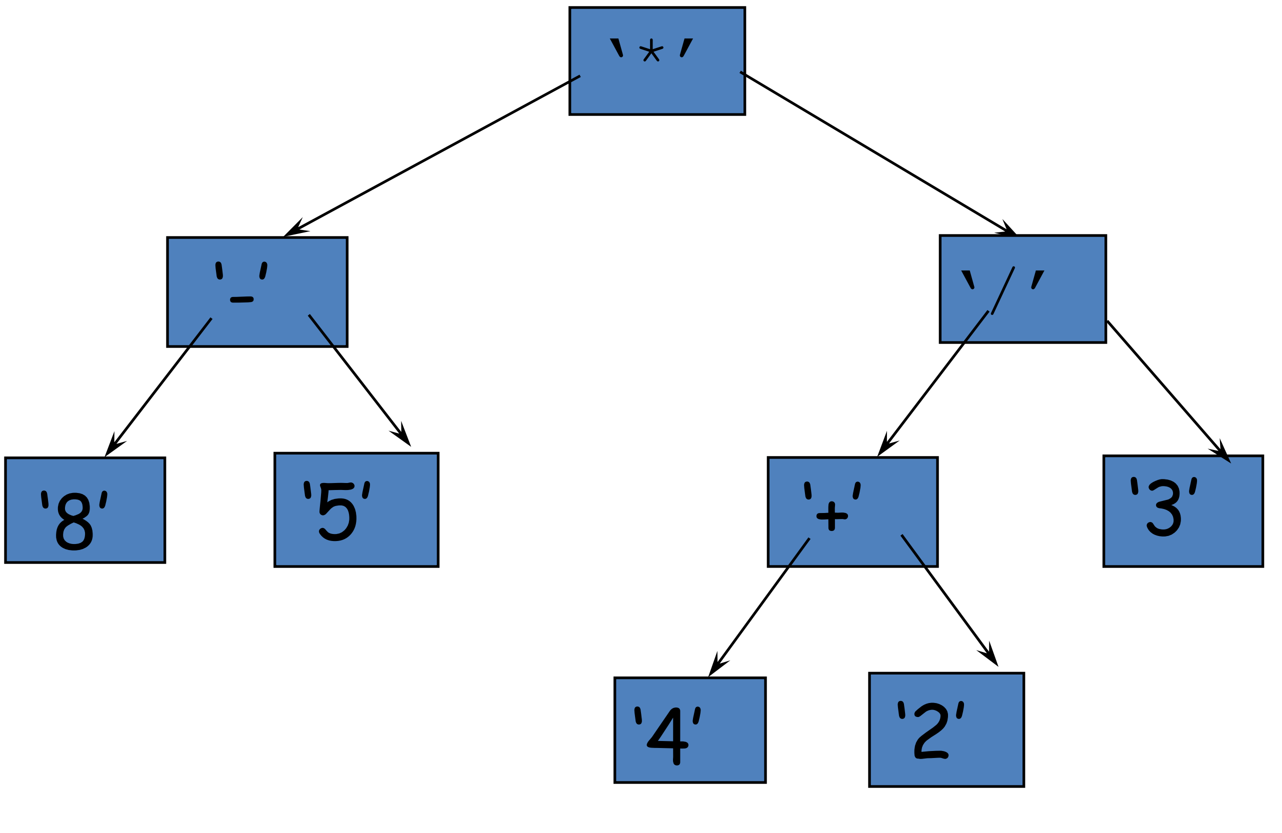
Binary Expression Tree

(Application of Binary Tree)

It is a special kind of binary tree in which:

1. **Each leaf node contains a single operand**
2. **Each nonleaf node contains a single binary operator**
3. **The left and right subtrees of an operator node represent subexpressions that must be evaluated before applying the operator at the root of the subtree.**

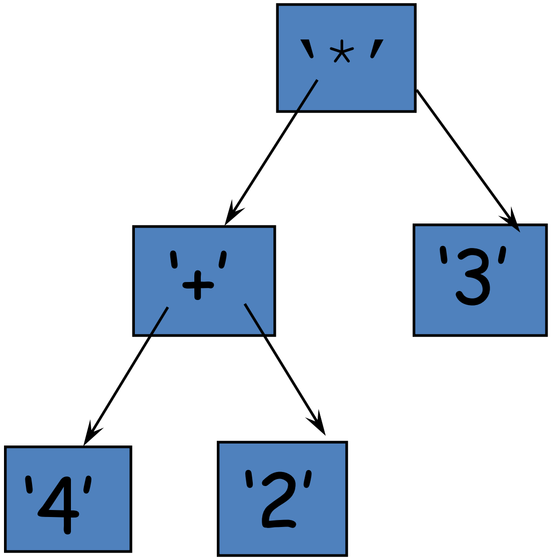
A Four-Level Binary Expression



Levels Indicate Precedence

- **The levels of the nodes in the tree indicate their relative precedence of evaluation (we do not need parentheses to indicate precedence).**
- **Operations at higher levels of the tree are evaluated later than those below them.**
- **The operation at the root is always the last operation performed.**

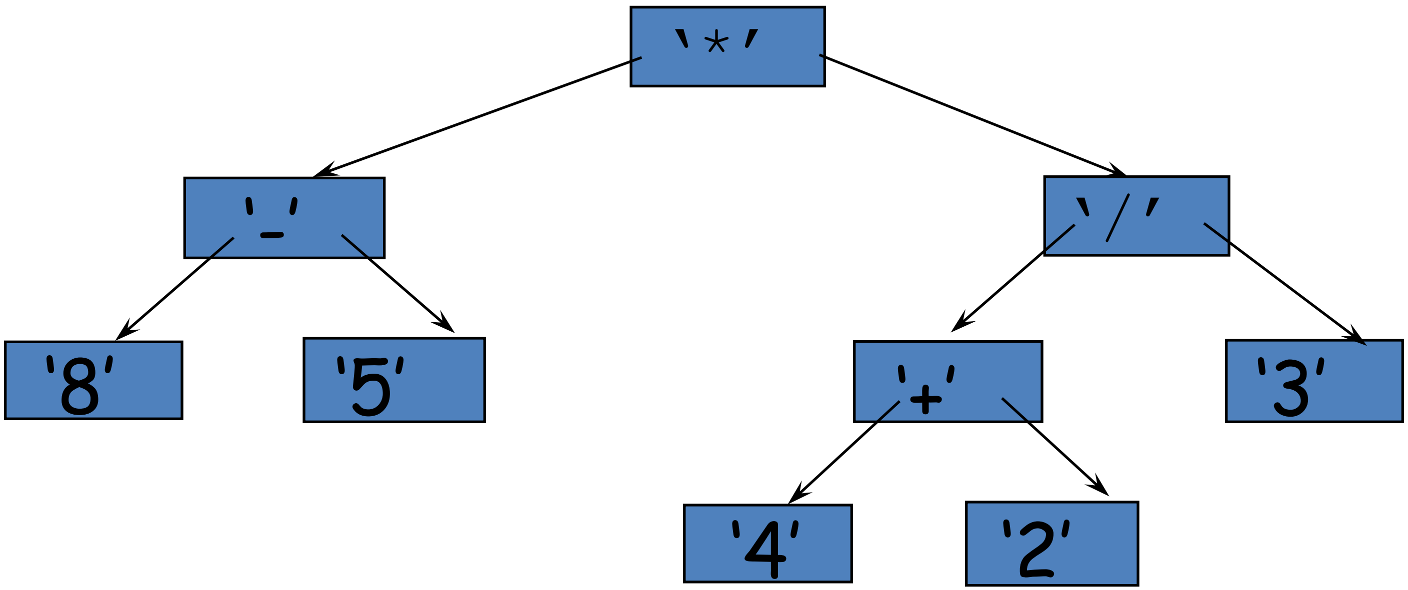
A Binary Expression Tree



What value does it have?

(4 + 2) * 3 = 18

Easy to generate the infix, prefix, postfix expressions (how?)



Infix: ((8 - 5) * ((4 + 2) / 3))

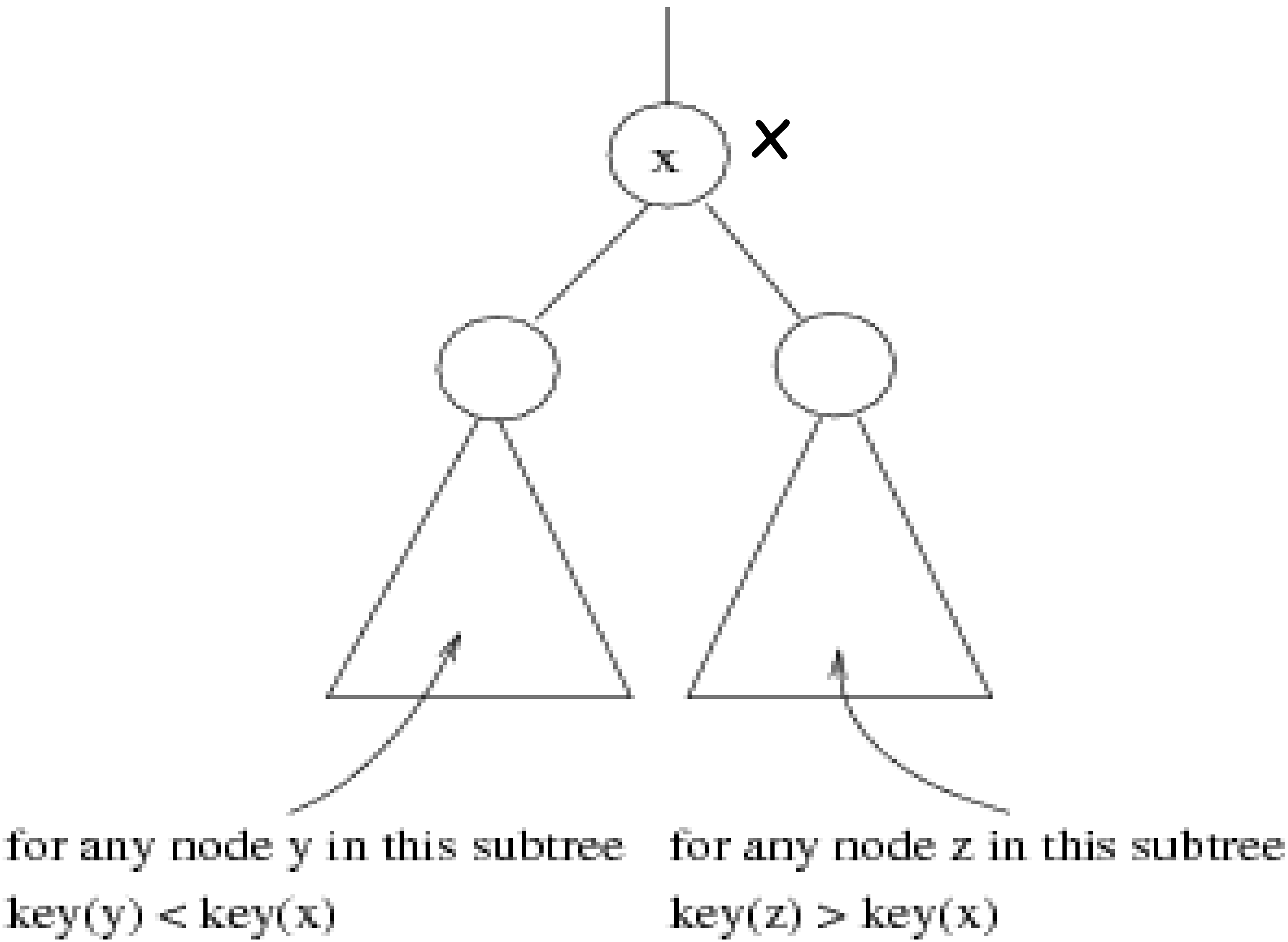
Prefix: * - 8 5 / + 4 2 3

Postfix: 8 5 - 4 2 + 3 / *

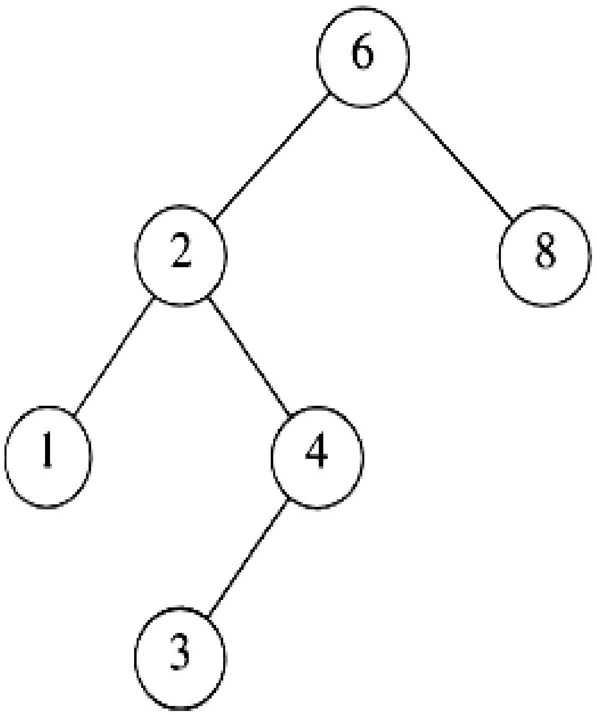
Binary Search Tree (BST)

Suppose **T** is a binary tree

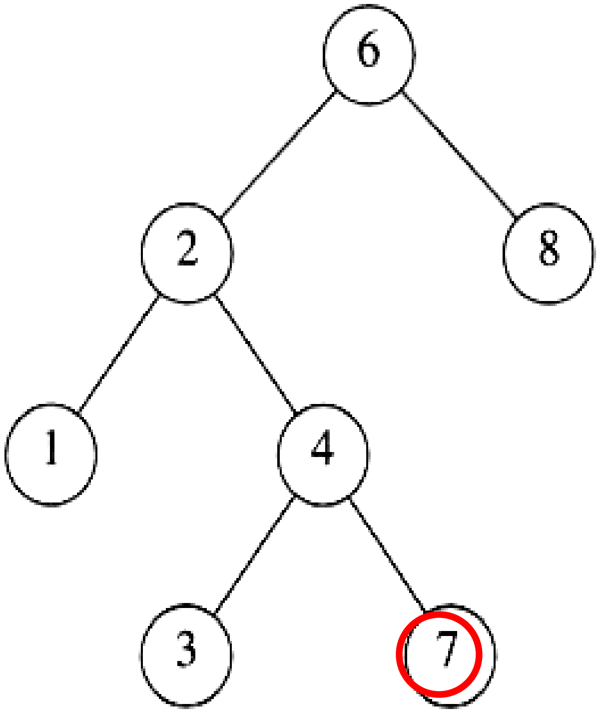
- Then **T** is called binary search tree if each node N of T has the following property
  - The value at N is **greater** than every value in the left sub-tree of N and is **less** than every value in the right sub-tree of N


x
x
for any node y in this subtree
key(y) < key(x)
for any node z in this subtree
key(z) > key(x)

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

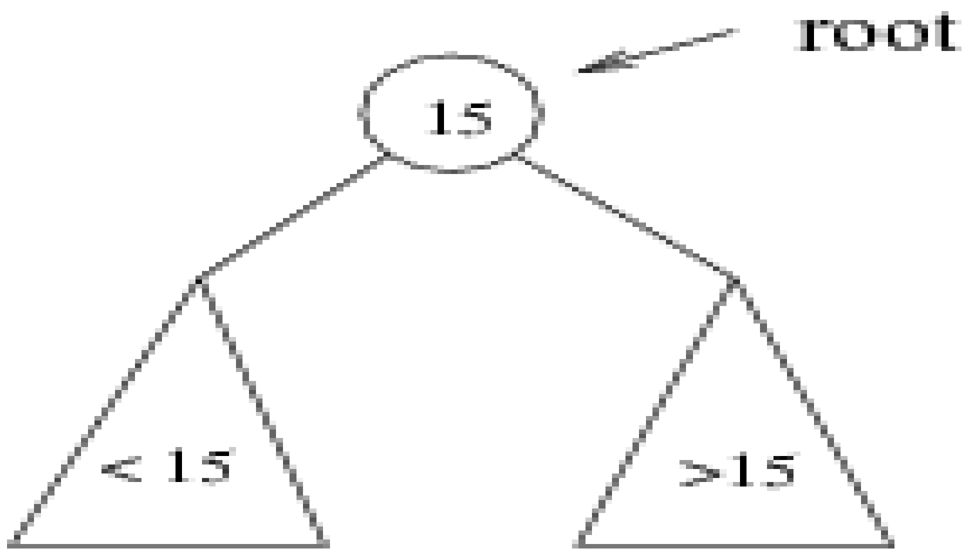
[b] Repeat Step (a) until one of the following occurs

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

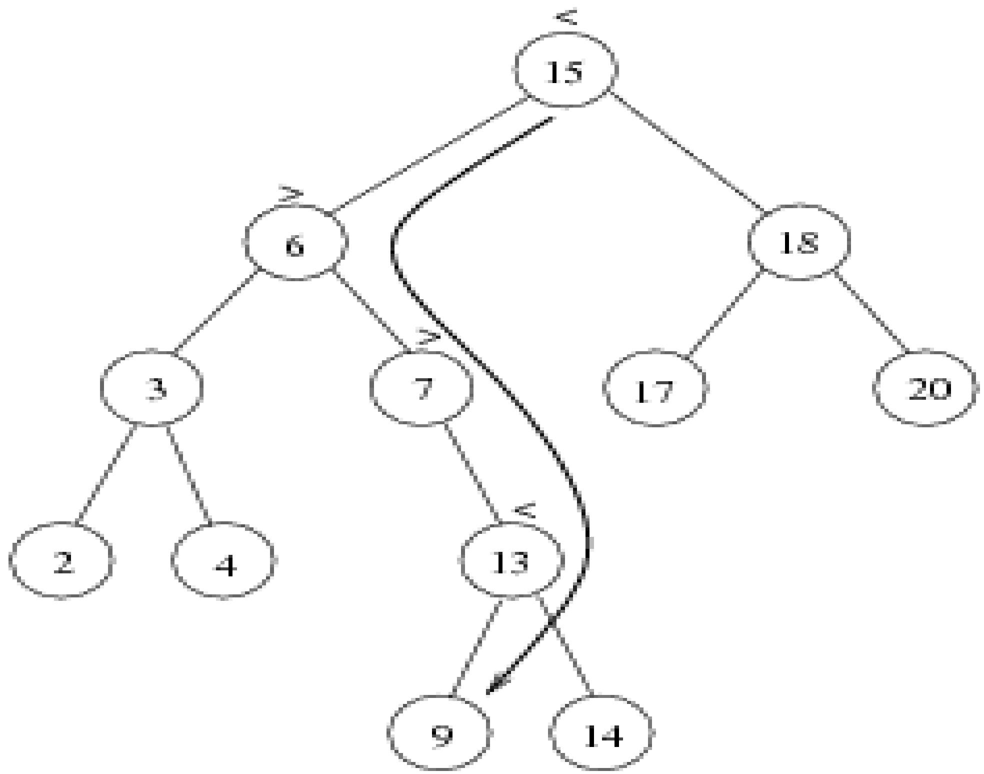
(ii) We meet an empty sub-tree, which indicates that search is unsuccessful and we insert ITEM in place of empty subtree

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.



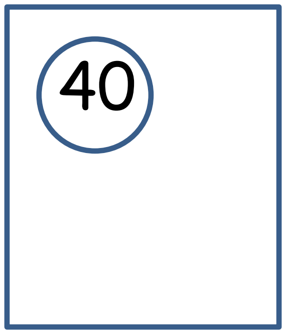
Example: Search for 9 ...



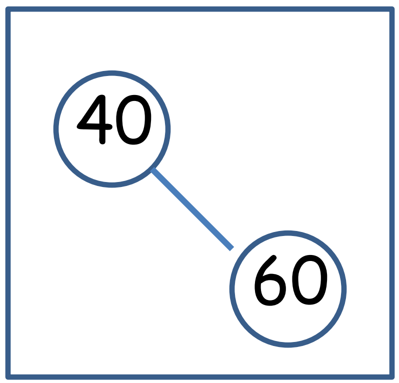
Search for 9:

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!

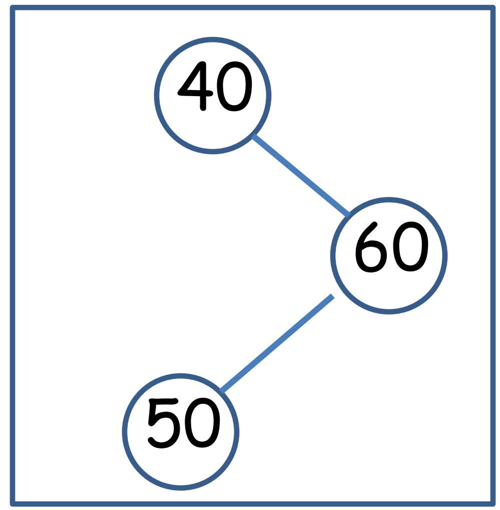
Insert 40, 60, 50, 33, 55, 11 into an empty BST



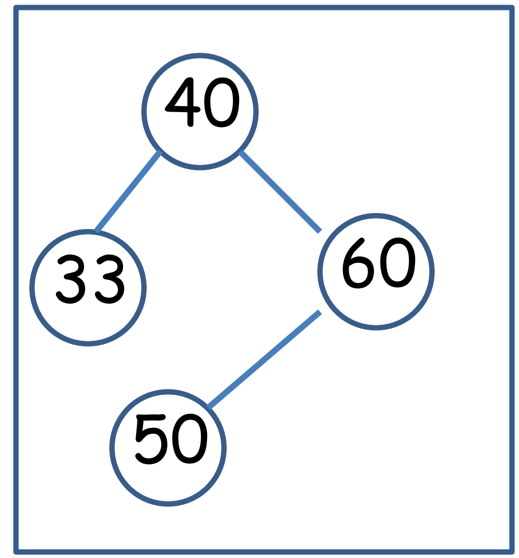
1. ITEM = 40



2. ITEM = 60

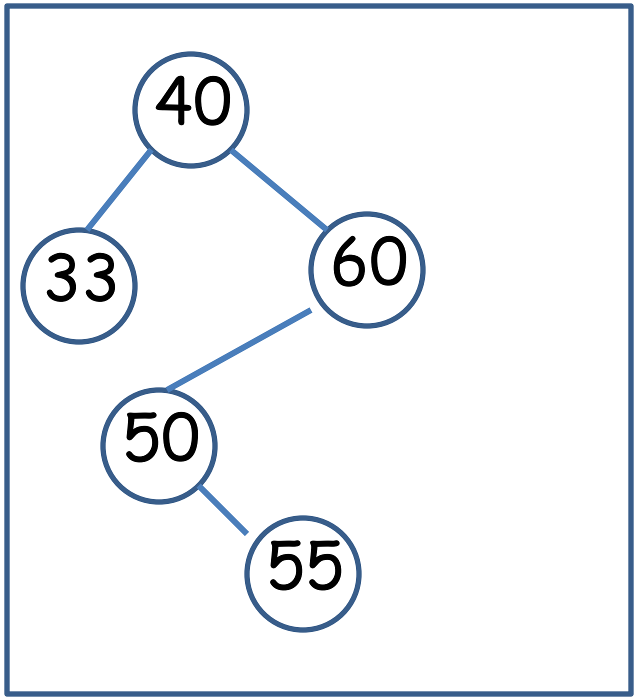


3. ITEM = 50

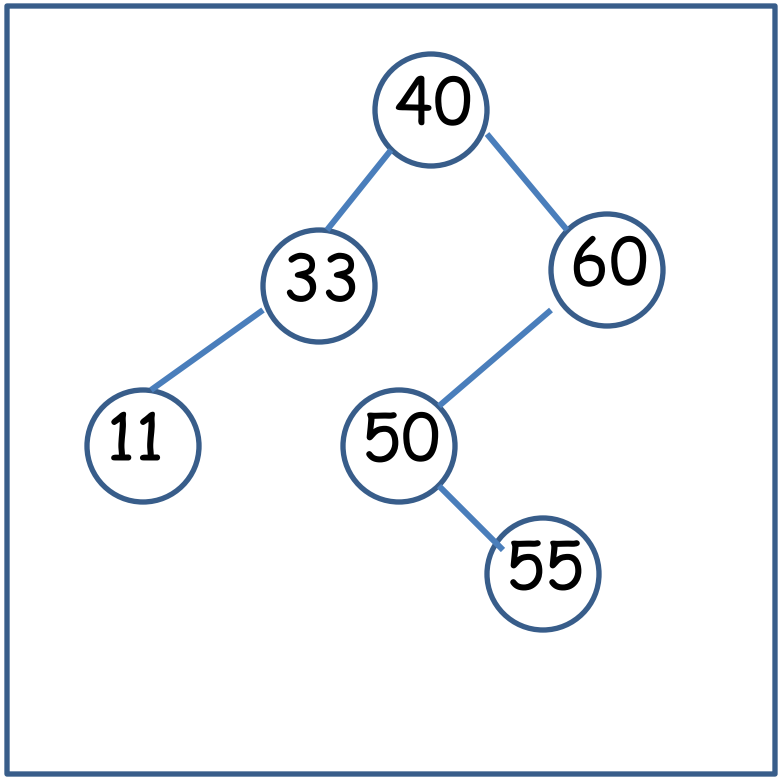


4. ITEM = 33

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.

Locating an ITEM

Three special cases:

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

[2] LOC $\neq$ NULL and PAR == NULL, ITEM is the root of T

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

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 ≠ 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

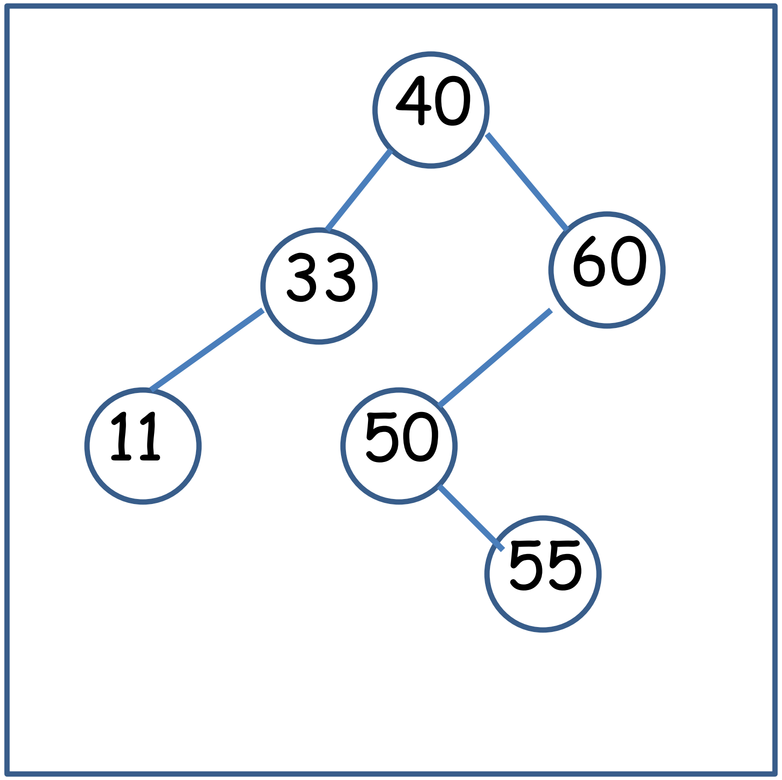
Else

Set SAVE = PTR, PTR = PTR→RIGHT

[7] [Search Unsuccessful]

Set LOC = NULL, PAR = SAVE

[8] Exit


40
33
60
11
50
55

A binary search Tree T is in memory and an ITEM of information is given. Algorithm to find the location LOC of ITEM in T or adds ITEM as a new node in T at location LOC.

[1] Call FIND(INFO, LEFT,RIGHT,ROOT,ITEM,LOC,PAR)

[2] If LOC $\neq$ NULL, then Exit

[3] [Copy the ITEM into the node NEW]

(a) Create a node NEW

(b) NEW$\rightarrow$INFO = ITEM

(c) Set LOC = NEW,

NEW$\rightarrow$LEFT = NULL,
NEW$\rightarrow$RIGHT = NULL

```
[4] [Add ITEM 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
```

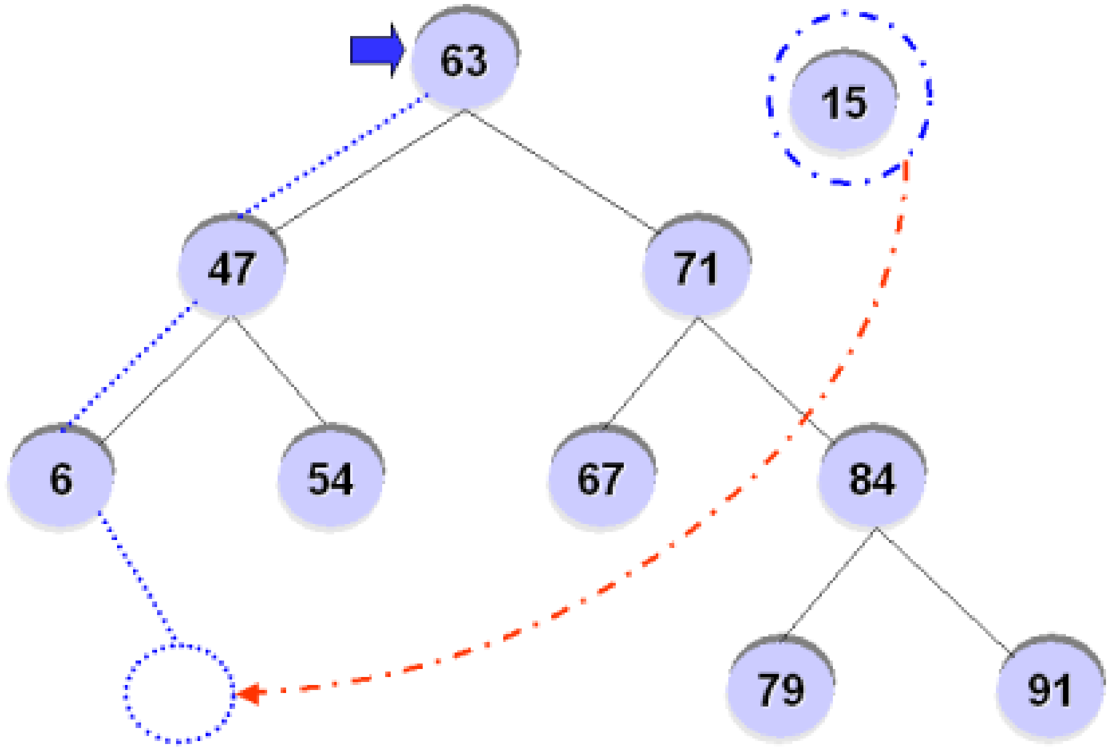
Binary Search Tree

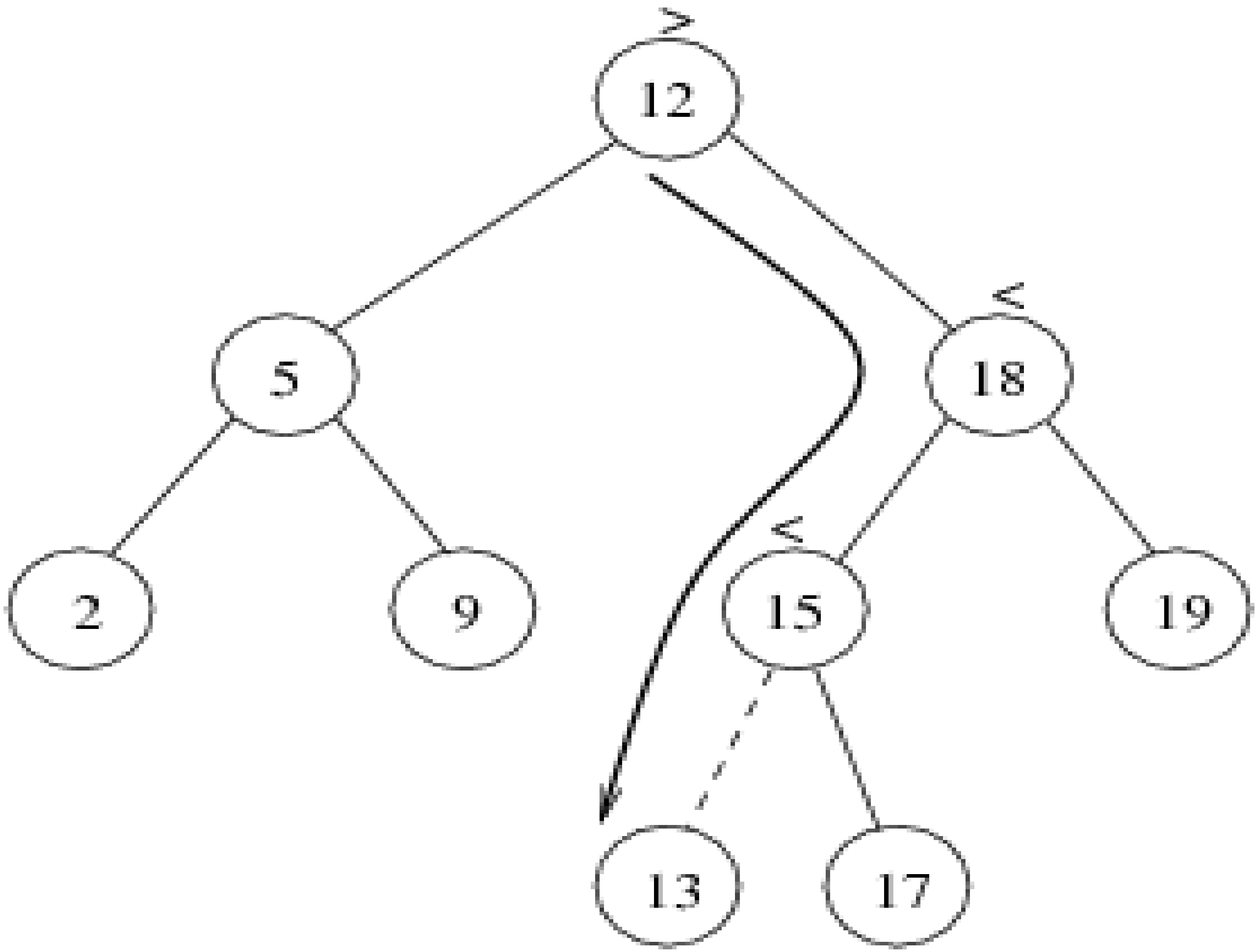
Insert Algorithm

- If value we want to insert < key of current node, we have to go to the left subtree
- Otherwise we have to go to the right subtree
- If the current node is empty (not existing) create a node with the value we are inserting and place it here.

Binary Search Tree

For example, inserting '15' into the BST?




>
12
<
5
18
<
2
9
15
19
13
17

Deletion from BST

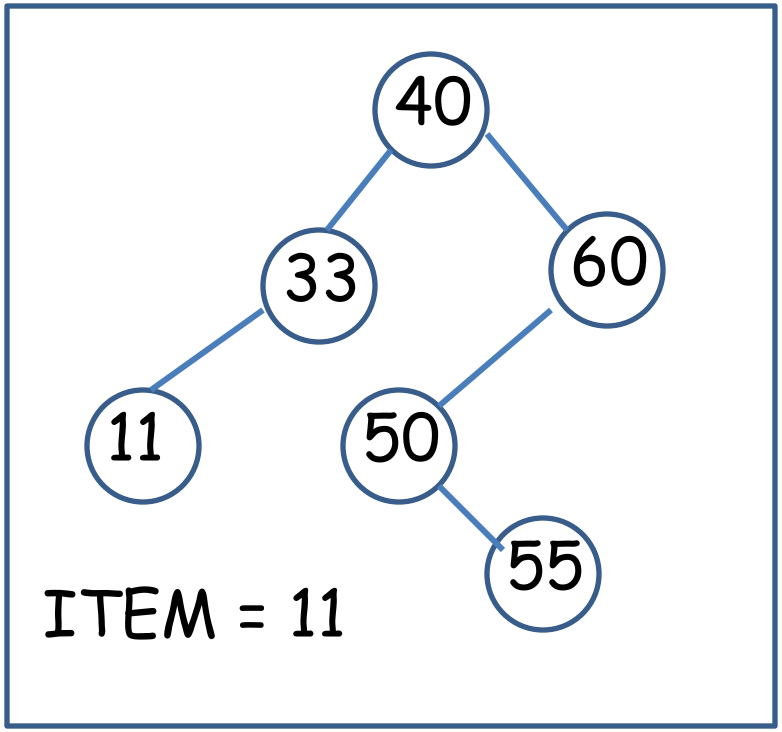
T is a binary tree. Delete an ITEM from the tree T

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

Deletion

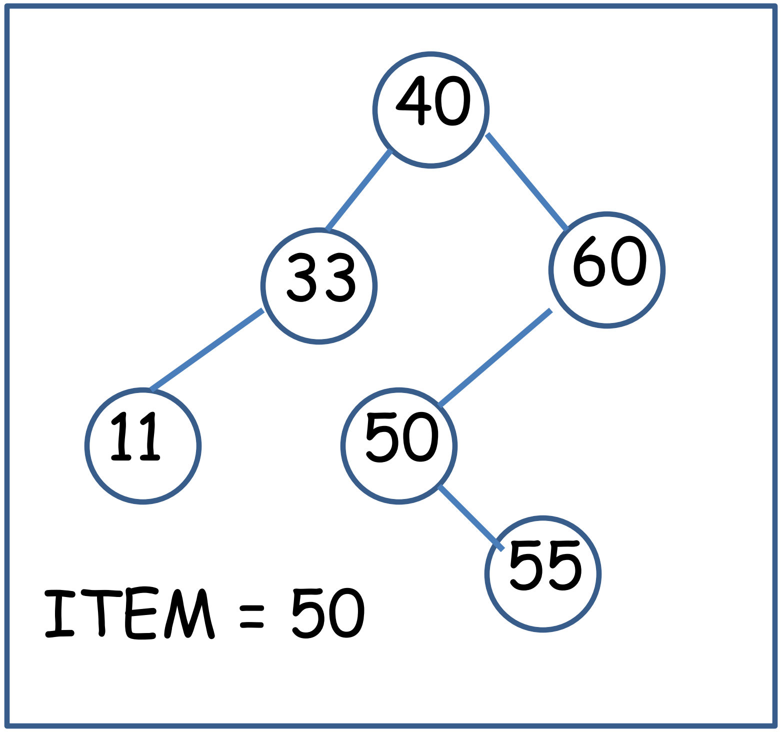
There are three cases in deletion

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



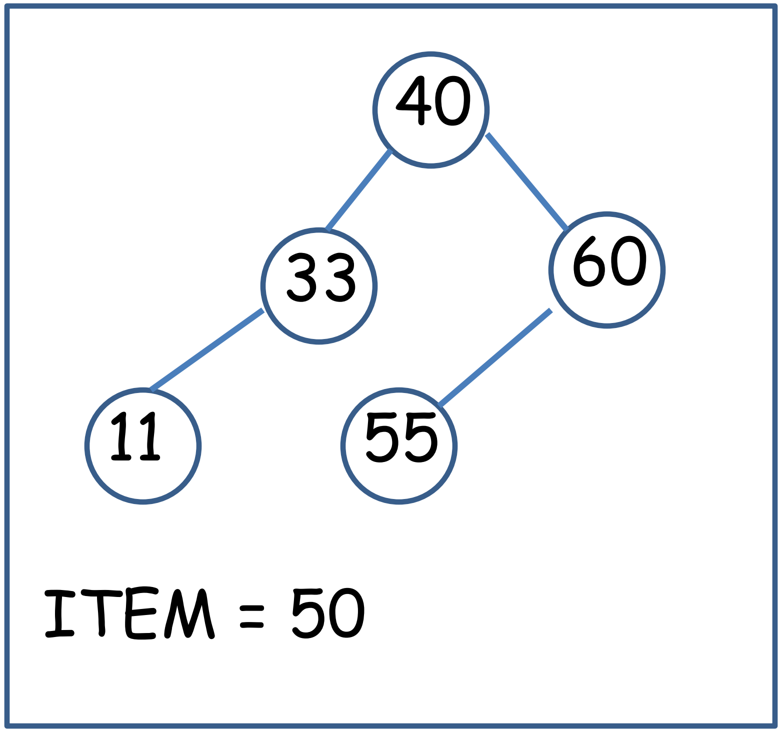
Deletion

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



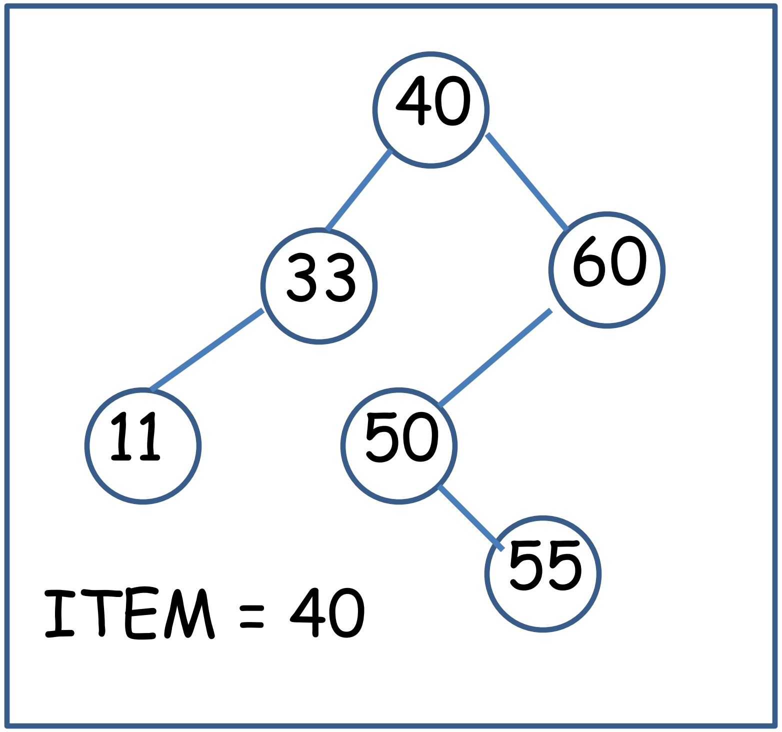
Deletion

Case 2.



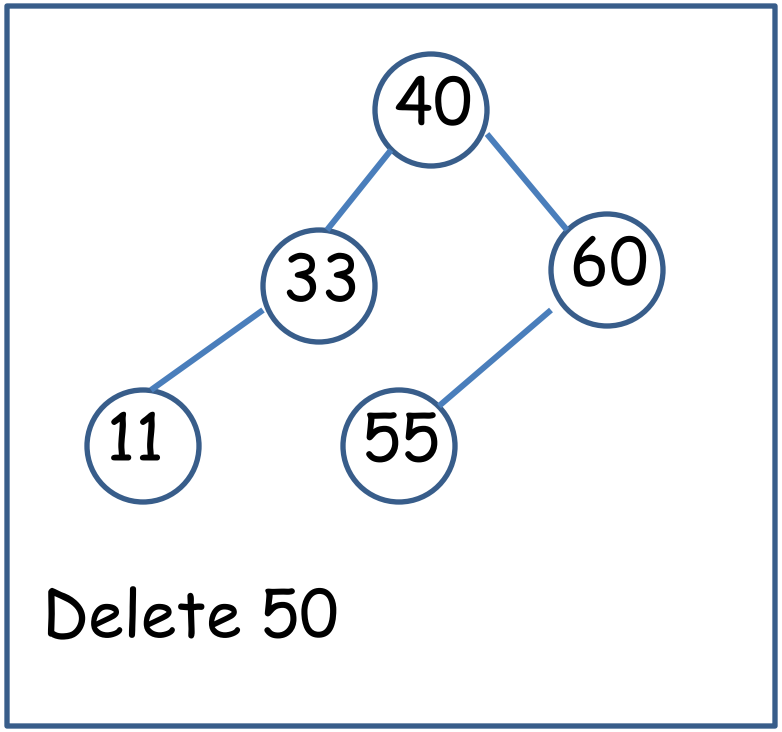
Deletion

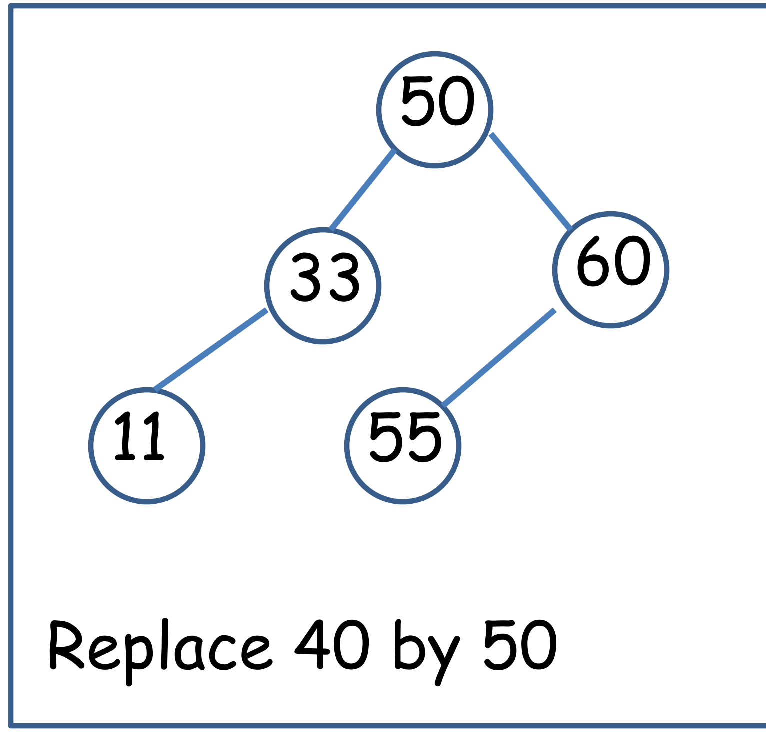
Case 3. N has two children. Let S(N) denote the in-order successor of N. Then N is deleted from the T by first deleting S(N) from T and then replacing node N in T by the node S(N)



Deletion

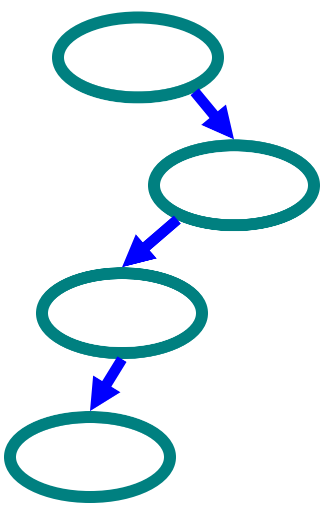
Case 3.



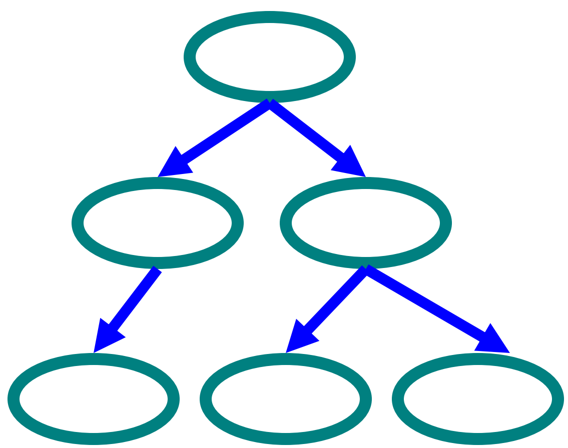


Types of Binary Trees

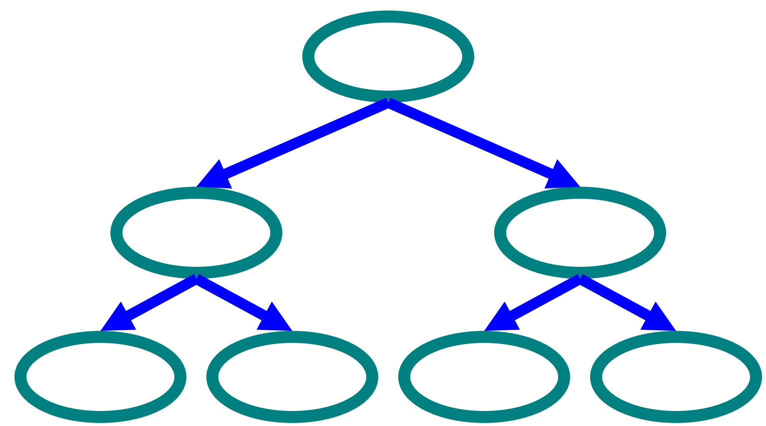
- Degenerate – only one child
- Balanced – mostly two children
- Complete – always two children



Degenerate binary tree



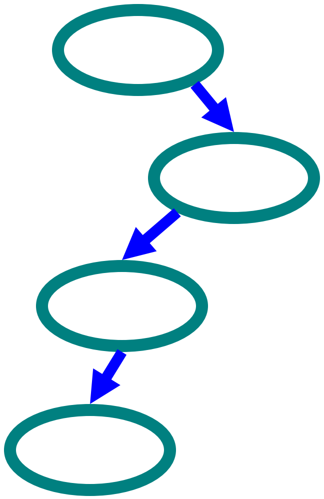
Balanced binary tree



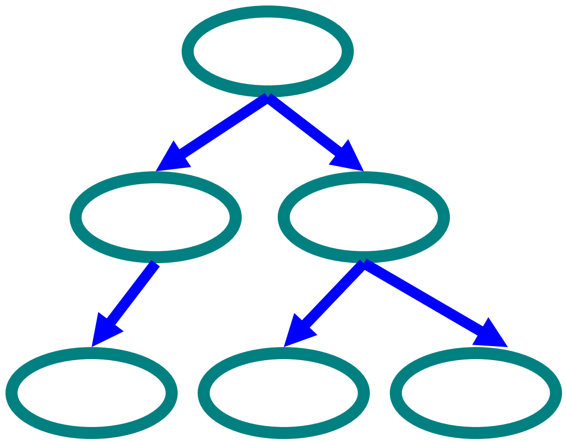
Complete binary tree

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