# Data Structure and Algorithms

Session-25

Dr. Subhra Rani Patra
SCOPE, VIT Chennai

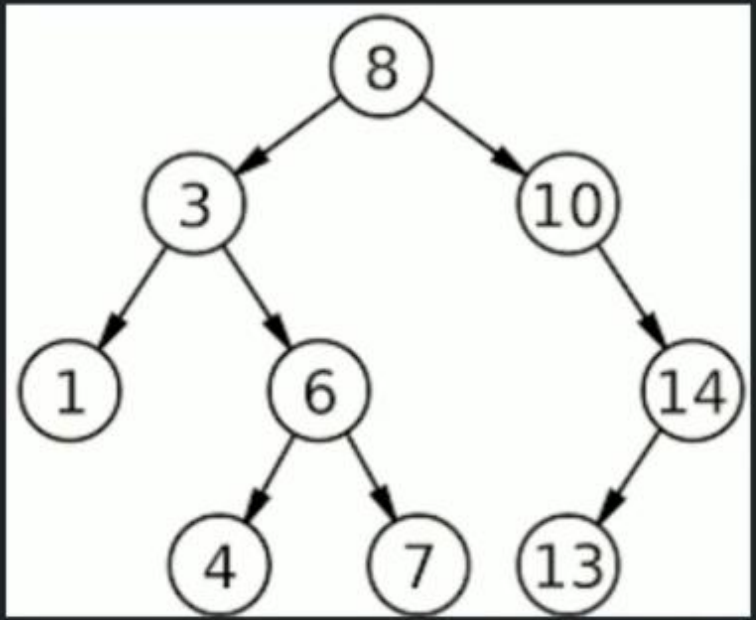
# What is BST ?

*Binary Search Tree (BST) is a Binary Tree in which all the nodes follows the below-mentioned properties:*

*✓ The left sub-tree of a node has a key less than or equal to its parent node's key.*

*✓ The right sub-tree of a node has a key greater than to its parent node's key*

*Sample BST:*



# Why should we learn BST ?

| Operation | Array | Linked List | Tree |
|---|---|---|---|
| Creation | $O(1)$ | $O(1)$ | Can we improve ? Let's see… |
| Insertion | $O(n)$ | $O(n)$ | |
| Deletion | $O(n)$ | $O(n)$ | |
| Searching | $O(n)$ | $O(n)$ | |
| Traversing | $O(n)$ | $O(n)$ | |
| Deleting entire Array/LinkedList/Tree | $O(1)$ | $O(1)$ | |
| Space Efficient ? | No | Yes | |

# Common operations of BST:

- ✓Creation of BST
- ✓Search for a value
- ✓Traverse all nodes
- ✓Insertion of a node
- ✓Deletion of a node
- ✓Deletion of  BST

# Algorithm - Creation of blank BST:

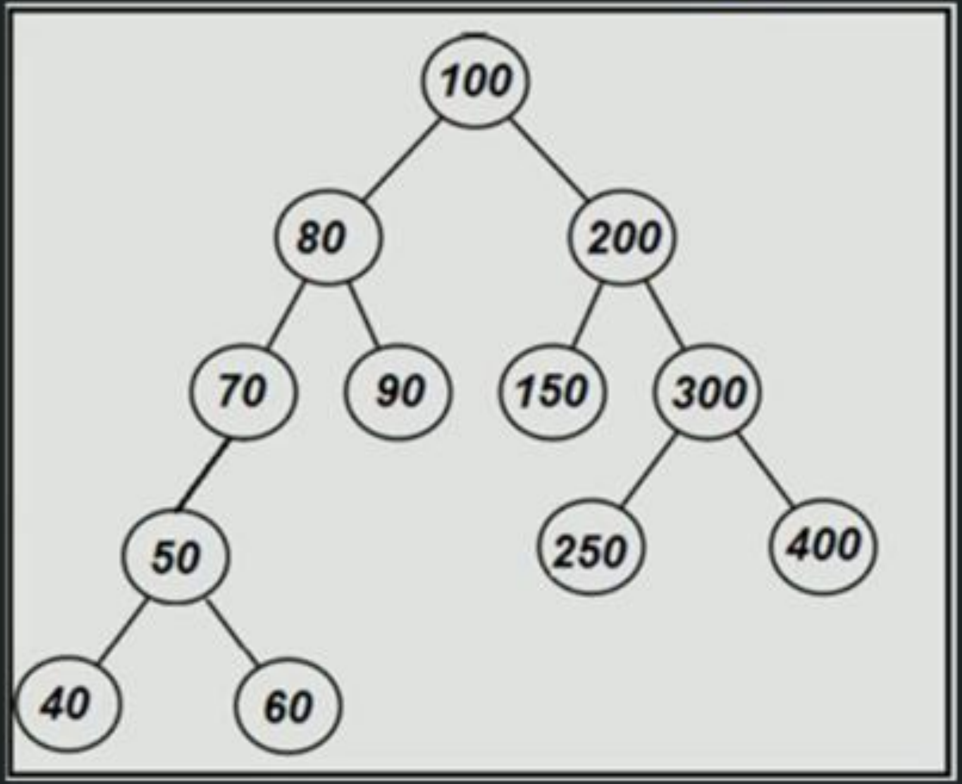
createBST()

Initialize Root with null

Time Complexity – O(1)

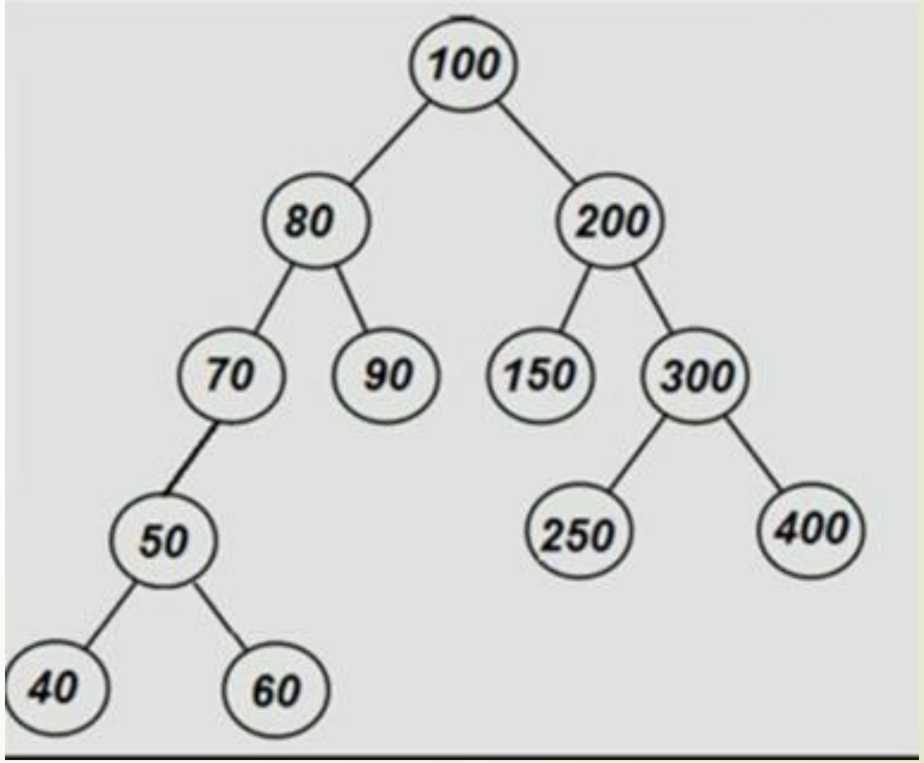
Space Complexity – O(1)

# Searching a node in BST:



# Algorithm - Searching a node in BST:

```
BST_Search (root, value):
  if (root is null)
      return null
  else if (root == value)
      return root
  else if (value < root)
      BST_Search (root.left, value)
  else if(value > root)
      BST_Search (root.right, value)
```



# Time & Space Complexity - Searching a node in BST:

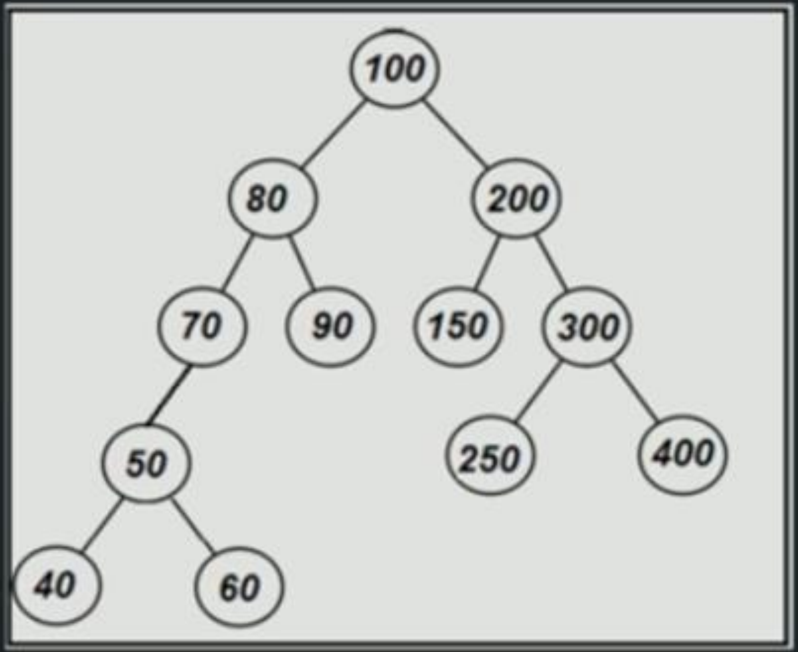
```
BST_Search (root, value) ---------------------------------------------- T(n)
  if (root is null) ---------------------------------------------------- O(1)
      return null  ---------------------------------------------------- O(1)
  else if (root == value) ---------------------------------------------- O(1)
      return root ------------------------------------------------------ O(1)
  else if (value < root) ----------------------------------------------- O(1)
       BST_Search (root.left, value) --------------------------------- T(n/2)
  else if(value > root) ------------------------------------------------ O(1)
      BST_Search (root.right, value) -------------------------------- T(n/2)
```

Time Complexity – O(log n)

Space Complexity – O(log n) (because of recursive call)

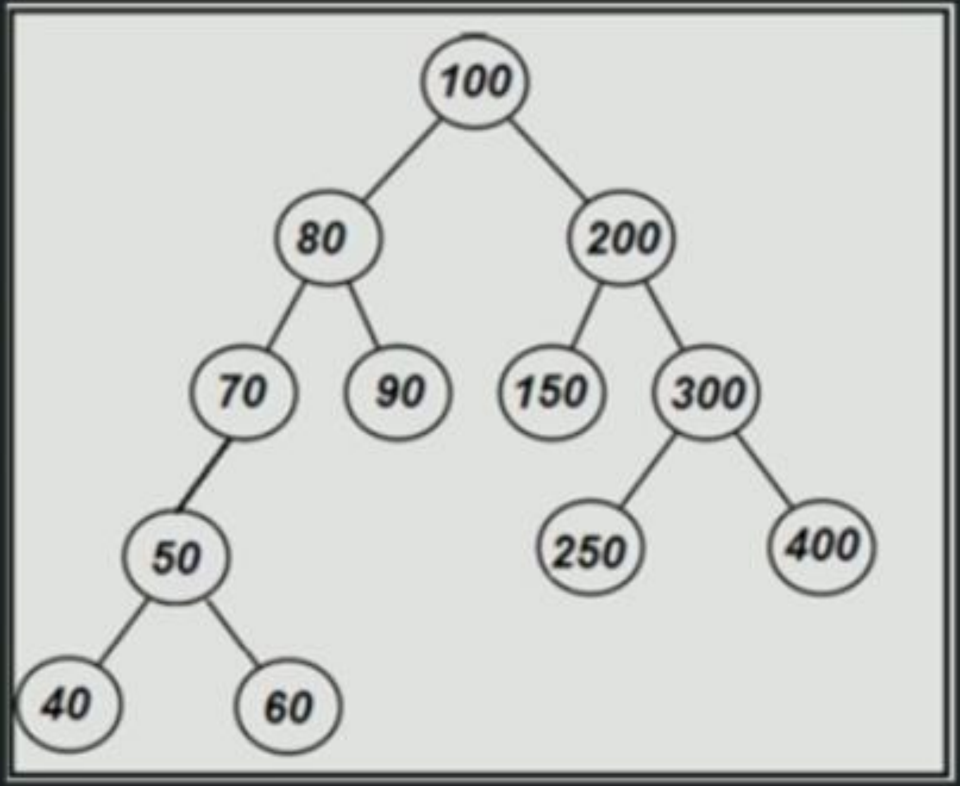
# Traversal of BST:

- ✓Depth First Search:
  - ✓ PreOrder Traversal
  - ✓ InOrder Traversal
  - ✓ PostOrder Traversal
- ✓Breadth First Search:
  - ✓ LevelOrder Traversal



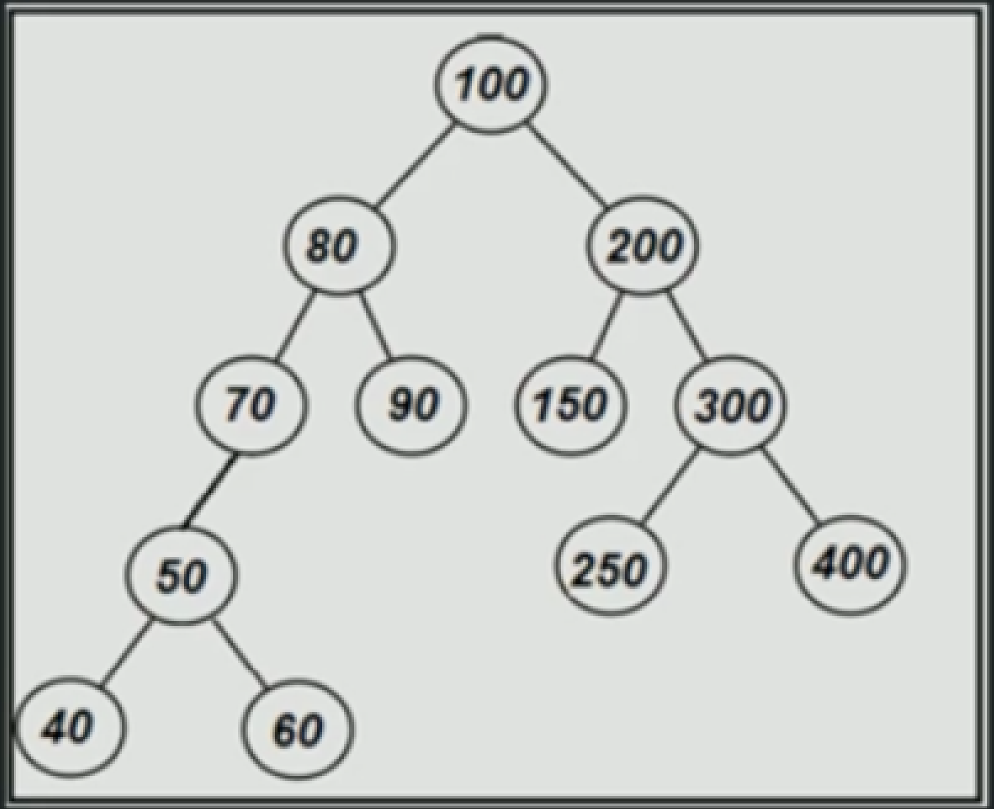
# Algorithm - Pre-Order Traversal of BST:

```
preorderTraversal(root)
  if (root equals null)
      return error message
  else
     print root
     preorderTraversal (root.left)
     preorderTraversal(root.right)
```



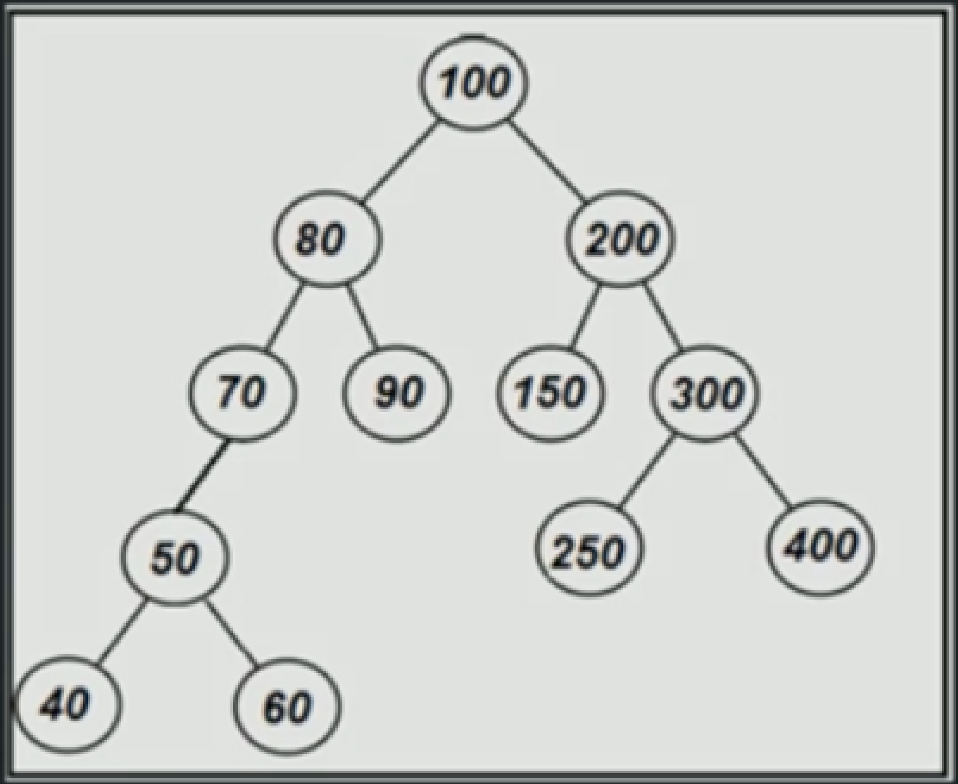
# Algorithm - 'In-Order Traversal' of BST:

```
inOrderTraversal (root)
  if (root equals null)
      return
  else
      inOrderTraversal(root.left)
      print root
      inOrderTraversal(root.right)
```



# Algorithm - ‘Post-Order Traversal’ of BST:

```
postOrderTraversal(root)
  if (root equals null)
      return
  else
      postOrderTraversal(root.left)
      postOrderTraversal(root.right)
      print root
```



# Algorithm - 'Level Order Traversal' of BST:

levelOrderTraversal(root)

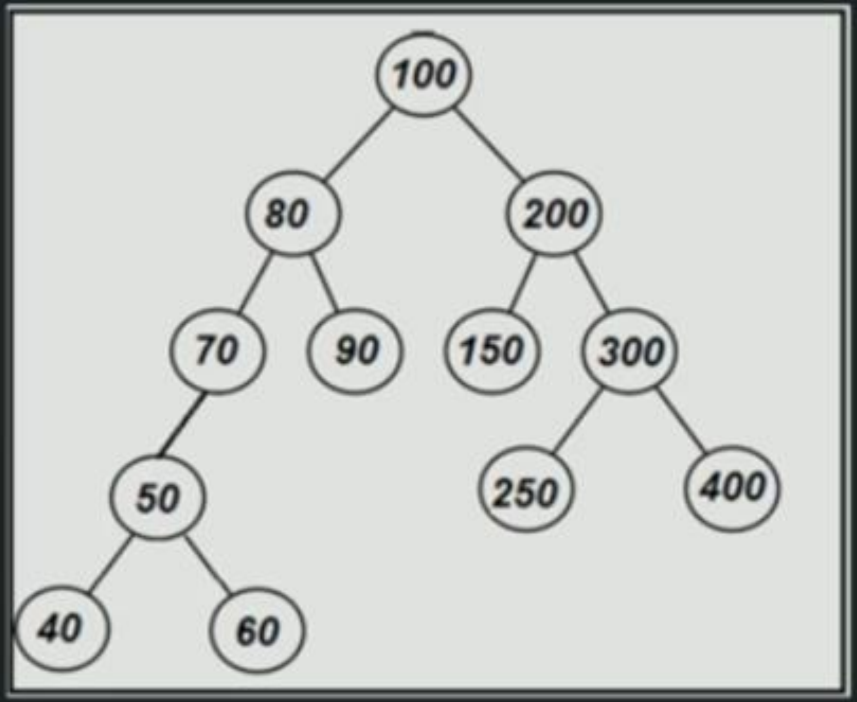
Create a Queue(Q)

enqueue(root)

While (Queue is not empty)

dequeue() and print

enqueue() the child of dequeued element

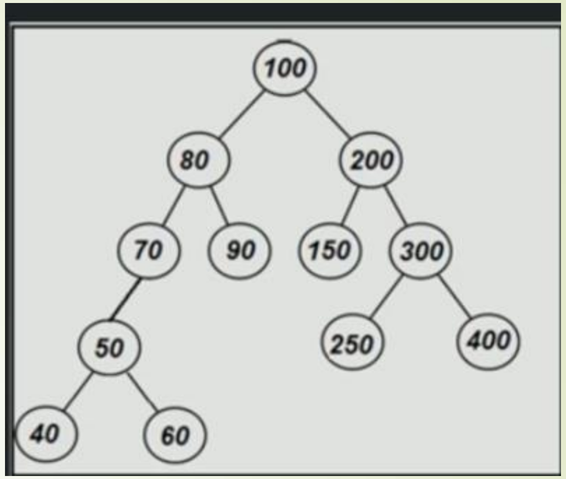


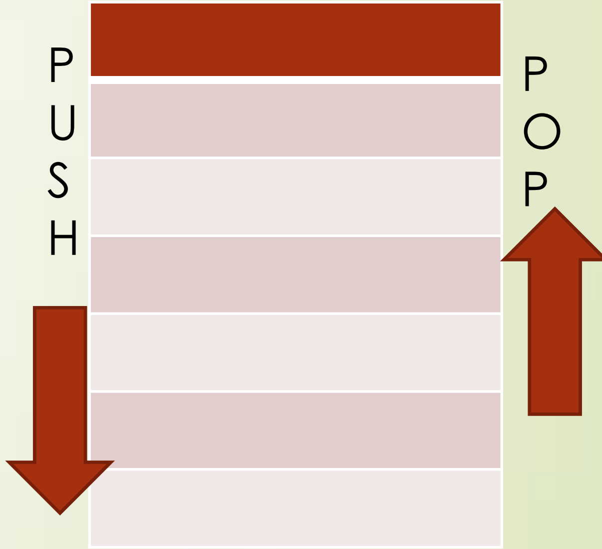
# Algorithm - Inserting a node in BST:

Cases:

1. BST is blank
2. BST is non-blank

```
BST_Insert (currentNode, valueToInsert)
    if (currentNode is null)
        create a node, insert 'valueToInsert' in it
    else if (valueToInsert <= currentNode 's value)
        currentNode.left = BST_Insert (currentNode.left, valueToInsert)
    else
        currentNode.right = BST_Insert (currentNode.right, valueToInsert)
    return currentNode
```



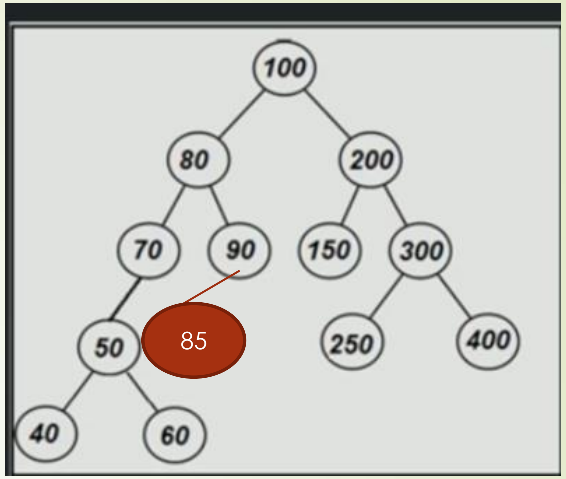



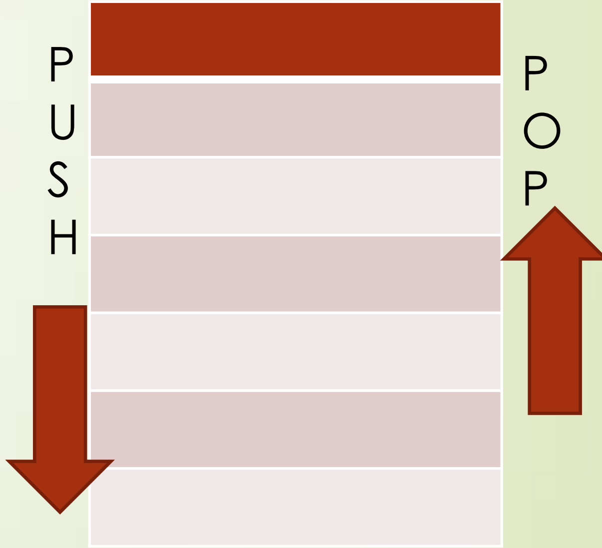
# Algorithm - Inserting a node in BST:

Cases:

1. BST is blank
2. BST is non-blank

```
BST_Insert (currentNode, valueToInsert)
    if (currentNode is null)
        create a node, insert 'valueToInsert' in it
    else if (valueToInsert  <= currentNode 's value)
        currentNode.left = BST_Insert (currentNode.left, valueToInsert)
    else
        currentNode.right = BST_Insert (currentNode.right, valueToInsert)
    return  currentNode
```



# *Time & Space Complexity - Inserting a node in B*

```
BST_Insert (currentNode, valueToInsert) ---------------------- T(n)
    if (currentNode is null) ------------------------------------------------ O(1)
        create a node, insert valueToInsert in it ---------------------------- O(1)
    else if (valueToInsert <= currentNode 's value) -------------------------- O(1)
        currentNode.left = BST_Insert (currentNode.left, valueToInsert) ------ T(n/2)
    else -------------------------------------------------------------------- O(1)
        currentNode.right = BST_Insert (currentNode.right, valueToInsert) ---- T(n/2)
    return currentNode ------------------------------------------------------ O(1)
```

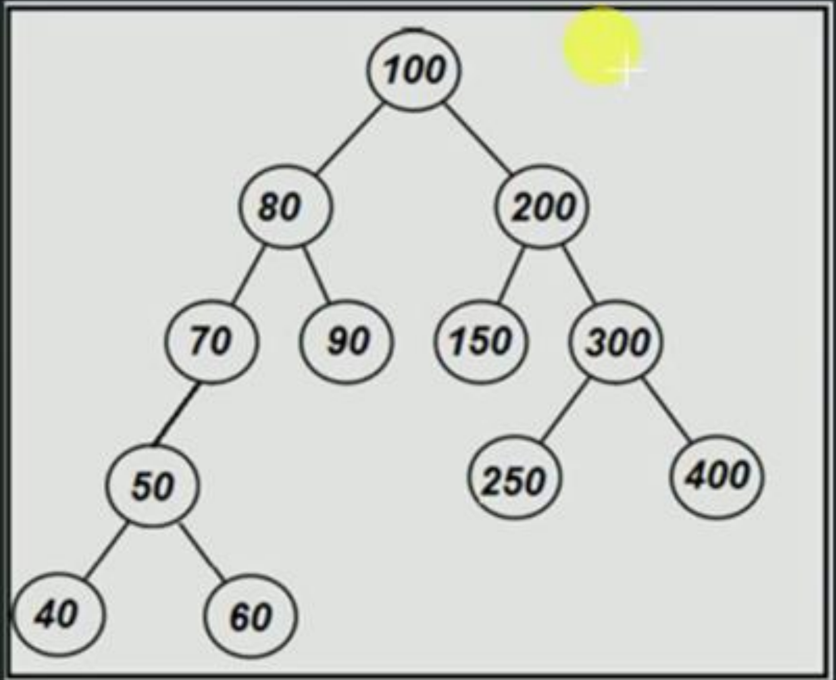
*Time Complexity – O(log n)*

*Space Complexity – O(log n)*

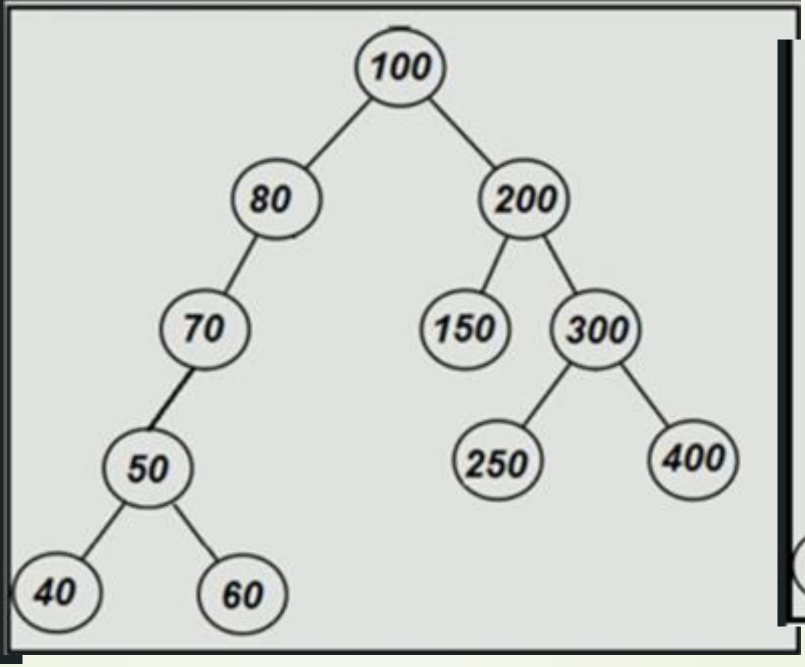
# Deletion of a node from BST:

- ✓ Node to be deleted is leaf node
- ✓ Node to be deleted is having 1 child
- ✓ Node to be deleted has 2 children

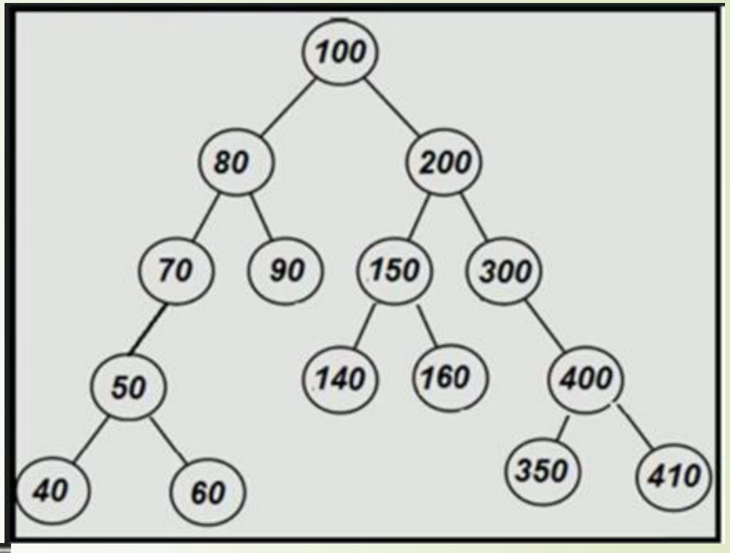
✓ Case#1 - Node to be deleted is leaf node

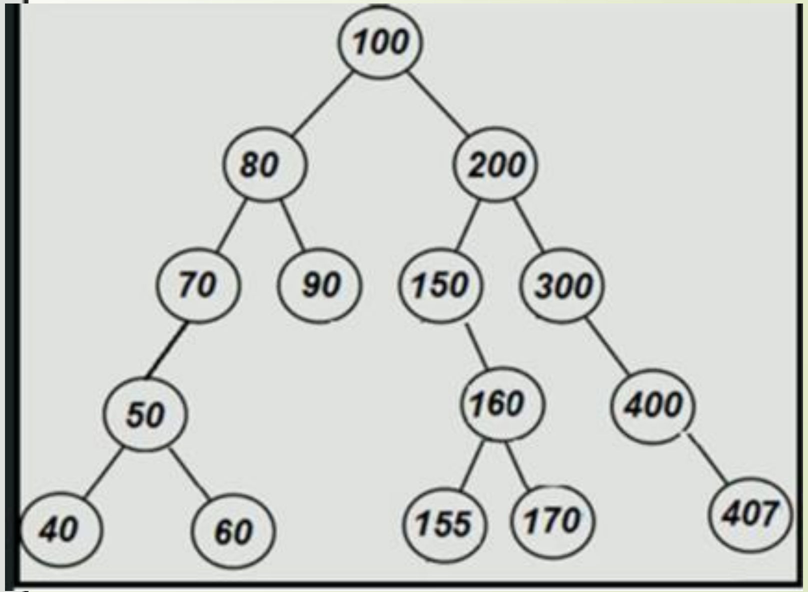



✓ Case#2 – Node to be deleted is having 1 child



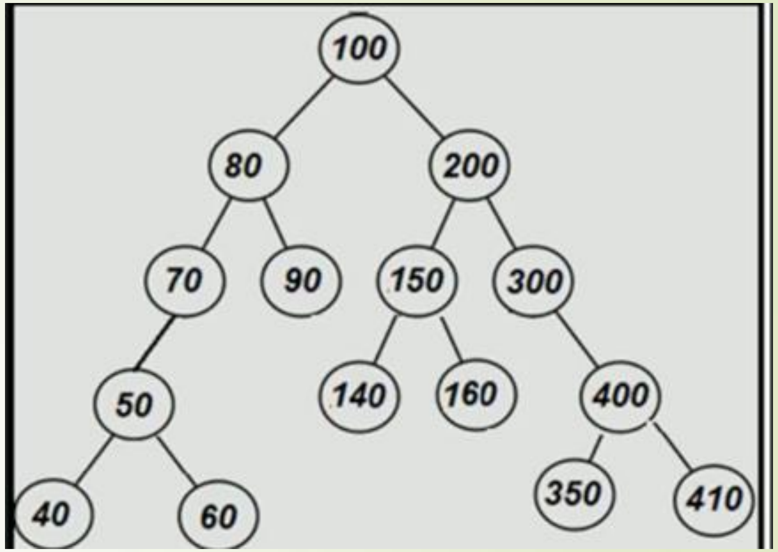

✓ Case#3 – Node to be deleted has 2 children



# Algorithm - Deletion of a node from BST:

```
deleteNodeOfBST (root, valueToBeDeleted):
    if (root == null) return null;
    if (valueToBeDeleted < root.Value)
          then root.left = deleteNodeOfBST (root.left, valueToBeDeleted)
    else if (valueToBeDeleted > root.value)
          then root.right = deleteNodeOfBST(root.right, valueToBeDeleted)
    else  // If currentNode is the node to be deleted
            if root have both children, then find minimum element from right subtree (Case#3)
                 replace current node with minimum node from right subtree and delete minimum node from right
            else if nodeToBeDeleted has only left child (Case#2)
                 then  root = root.Left()
            else if nodeToBeDeleted has only right child (Case#2)
                  then root = root.Right();
             else // if nodeToBeDeleted do not have child (Case#1)
              root = null;
     return root;
```



# Time & Space Complexity

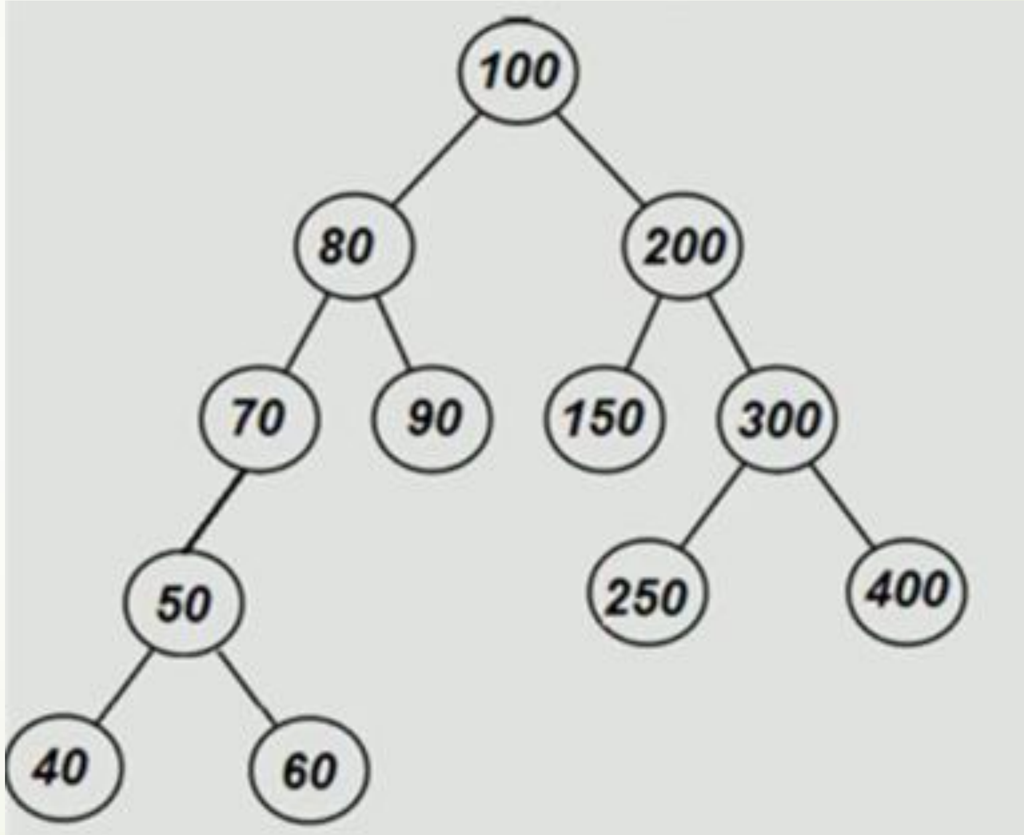
```
deleteNodeOfBST (root, valueToBeDeleted) ------------------ T(n)
    if (root == null) return null; ----------------------------------------- O(1)
    if (valueToBeDeleted < root.Value) ------------------------------------- O(1)
        then root.left = deleteNodeOfBST (root.left, valueToBeDeleted) ----- T(n/2)
    else if (valueToBeDeleted > root.value) -------------------------------- O(1)
        then root.right = deleteNodeOfBST(root.right, valueToBeDeleted) ---- T(n/2)
    else  // If currentNode is the node to be deleted ---------------------- O(1)
          if root have both children, then find minimum element from right subtree ---- O(log n)
              replace current node with minimum node from right subtree and delete minimum node from right ---- O(1)
          else if nodeToBeDeleted has only left child ---------------------- O(1)
              then  root = root.Left() ------------------------------------- O(1)
          else if nodeToBeDeleted has only right child --------------------- O(1)
                 then root = root.Right(); --------------------------------- O(1)
           else // if nodeToBeDeleted do not have child (Leaf node) -------- O(1)
                root = null; ----------------------------------------------- O(1)
     return root ----------------------------------------------------------- O(1)
```

*Time Complexity* – O(log n)

*Space Complexity* – O(log n)

# Algorithm - Deletion of entire BST:

```
DeleteBST()
    root = null
```



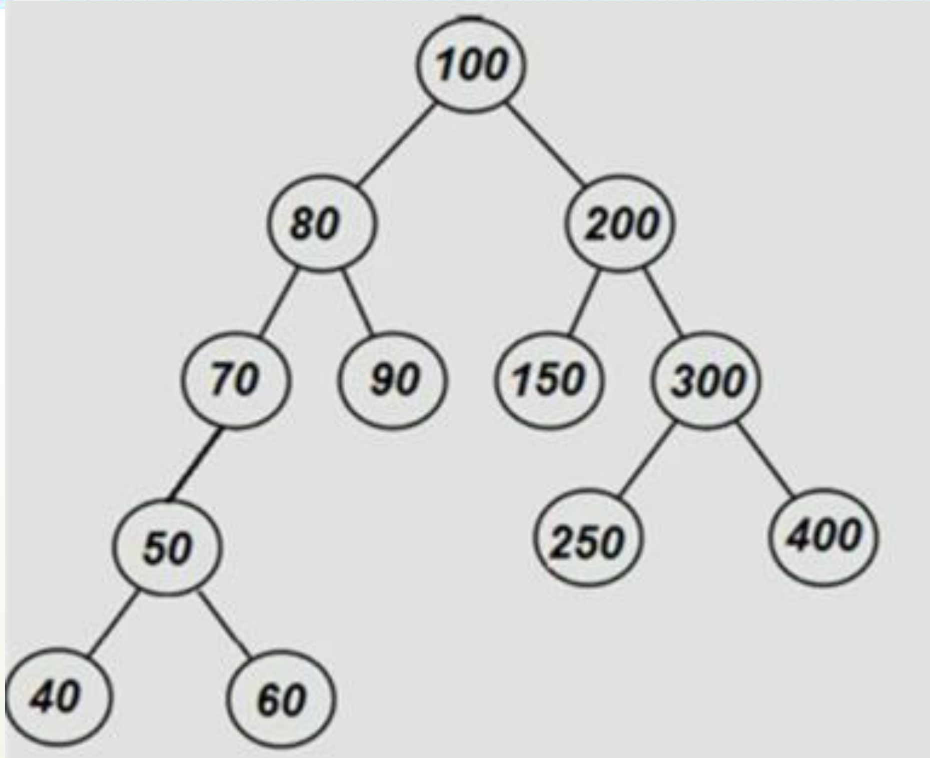
# Find Minimum and Maximum of a BST

**TREE-MINIMUM (x)**

```
while left[x] ≠ NIL do
    x ← left [x]
return x
```

**TREE-MAXIMUM (x)**

```
while right[x] ≠ NIL do
    x ← right [x]
return x
```




Thank you