Book

A Simplified Approach
to
Data Structures

Prof.(Dr.)Vishal Goyal, Professor, Punjabi University Patiala
Dr. Lalit Goyal, Associate Professor, DAV College, Jalandhar
Mr. Pawan Kumar, Assistant Professor, DAV College, Bhatinda

Shroff Publications and Distributors

Edition 2014

BINARY SEARCH TREE(**BST**)

Contents for Today's Lecture

- Binary Search Tree
- Operations on binary search tree
  - Searching a particular element in BST
  - Insertion of an element
  - Deletion of an element
  - Finding the smallest element
  - Finding the largest element

Binary Search Tree

- Binary search tree(BST) is a very important subclass of binary trees.
- In binary trees data is not ordered in some logical order But in BST, data is managed in such a logical way that it can be retrieved efficiently when required**.**
- A binary search tree is a binary tree in which node containing the data has the following constraints:

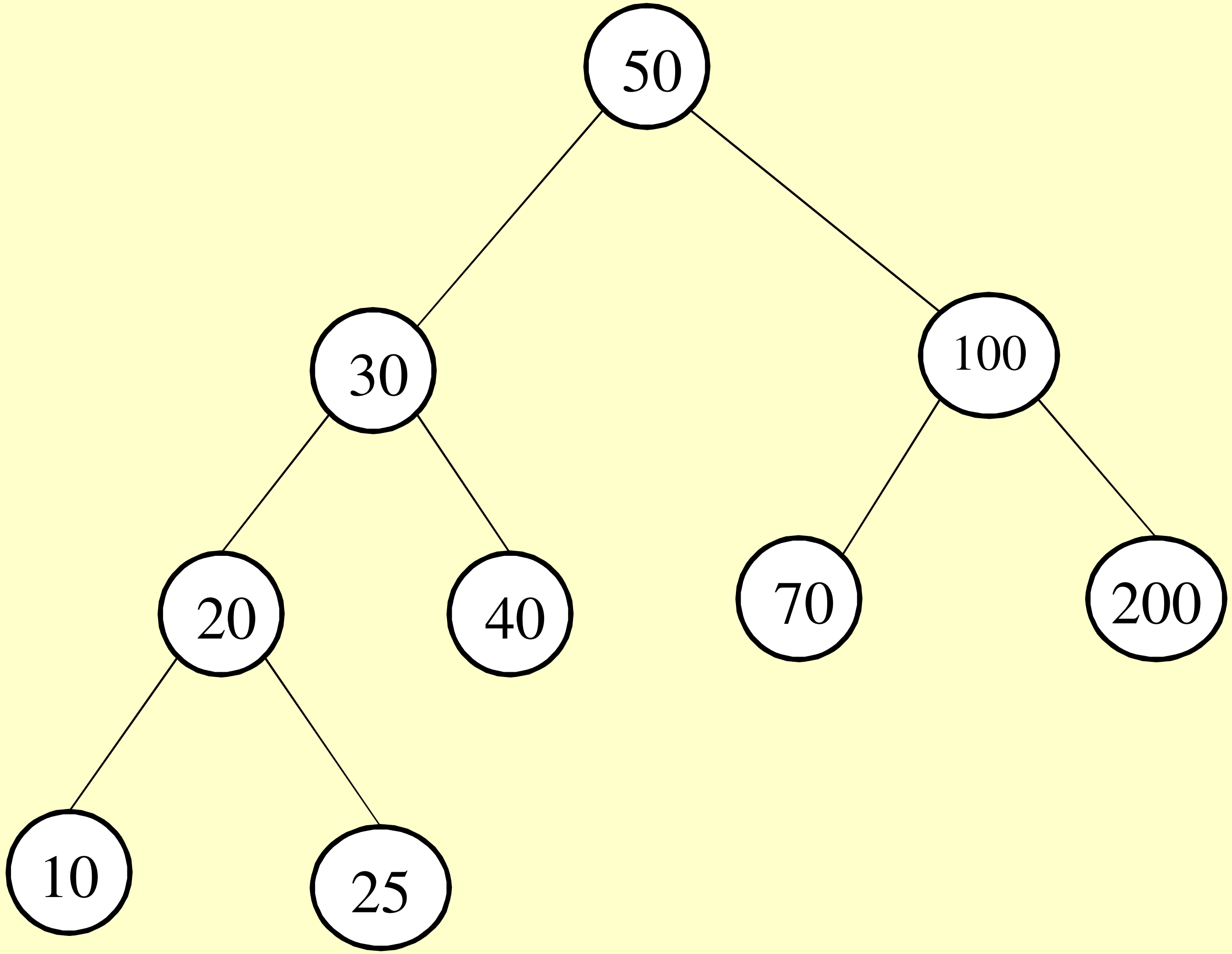
➢Each data element in the left subtree is less than its root element.

➢Each data element in the right subtree is greater than or equal to its root element.

➢Both the left and right subtree of the root will be again a BST.

Binary Search Tree(continued)

The binary tree shown is binary search tree.



When this BST is traversed it produces a sorted list of data elements.

Operations on Binary Search Tree

Various operations that can be performed on binary search tree are:

- Searching a particular key element.
- Inserting an element.
- Deletion of an element.
- Finding the smallest element.
- Finding the largest element.

Searching of a particular key value in BST

- To search particular element in a binary search tree, start at the root by comparing the desired element with the value stored at root.
- If both are same , stop the search.
- Otherwise follow the left or right subtree depending on whether the given element is less than or larger than the element stored at root node.
- This procedure is repeated recursively until we find the desired element
- If not found then conclude that element is not present in the binary search tree.

ALGORITHM to search a particular value in BST

BSTSearch(Root,Item,Position,Parent)

Step1: If **Root=Null** Then

set **Position** = **null**

set **Parent = null**

Return

[End If]

Step 2: **Pointer=Root** And **Pointer P = Null**

Step 3: Repeat Step 4 While **Pointer != Null**

Step 4: If **Item = Pointer→Info** Then

set **Position** = **Pointer**

set **Parent = PointerP**

Return

Step 5: Else If **Item**<Pointer→**Info** Then

ALGORITHM(continued)

set **PointerP** = **Pointer**

set **Pointer = Pointer→Left**

Else

set **PointerP** = **Pointer**

set **Pointer = Pointer→Right**

[End If]

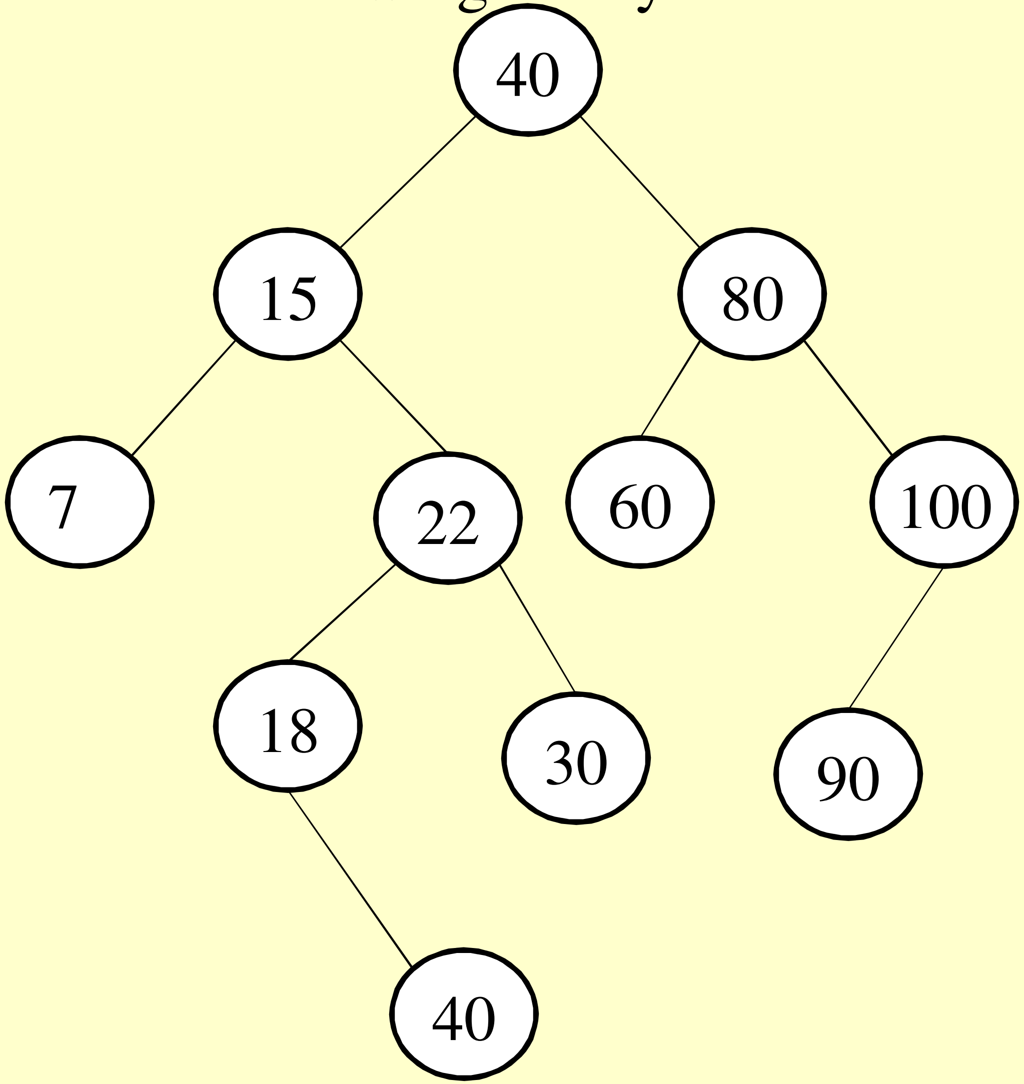
[End Loop]

Step 5: Set **Position =Null** And **Parent = Null**

Step 6: Return

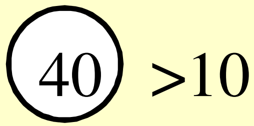
Inserting of a particular key value in BST

Consider the following binary tree :

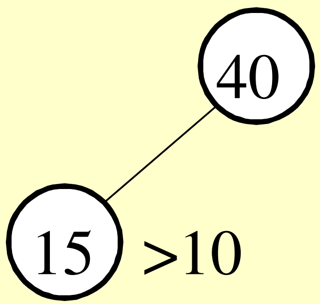


Inserting of a particular key value in BST

While inserting a new element into the binary search tree, the properties of the BST must be preserved
so that the tree remains a binary search tree even after insertion.
Consider an Item=10 is to be inserted .
Firstly compare 10 with root value i.e 40

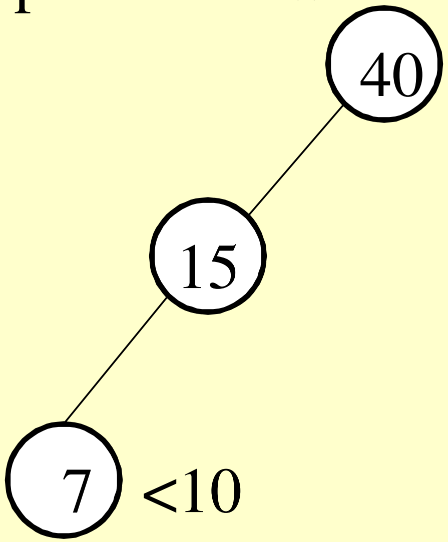


As 10 is less than root element (40) so we will proceed towards leftsubtree .
Now 10 is compared with root of left subtree. i.e 15



Inserting of a particular key value in BST

As 10<15 so proceed towards left subtree.

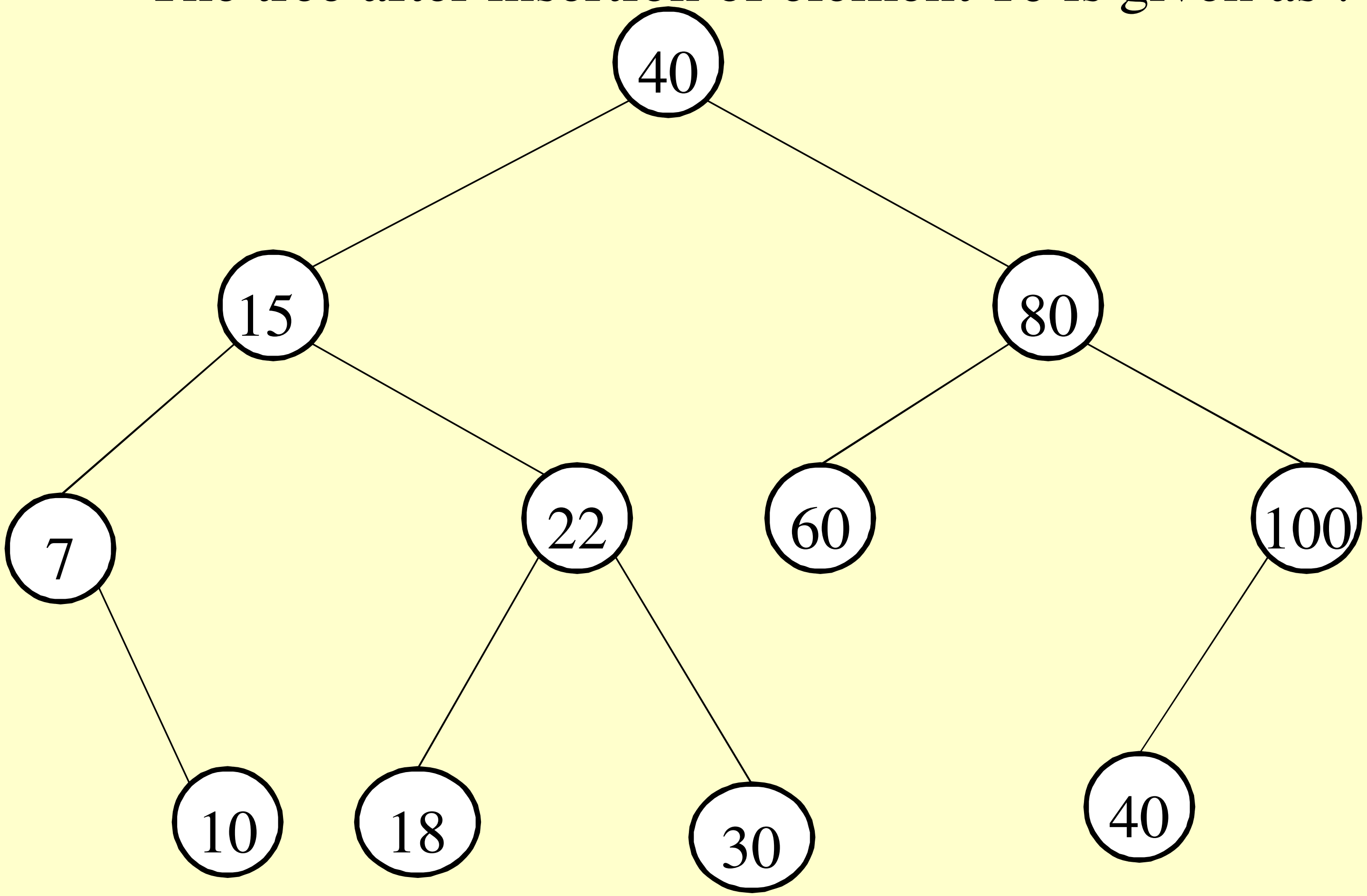


Now 10 is compared with 7

As 7<10 and there is no right subtree so insert the new element in right node as shown:

Inserting of a particular key value in BST

The tree after insertion of element 10 is given as :-



Algorithm to insert a given element

Step 1: If **Free = Null** Then

Print: "No space is available for the node to insert"

Exit

Else

Allocate memory to new node for insertion

(**New** = **Free** And **Free = Free**→ Right)

Set **New→Info= Item**

Set New→left= Null And **New→Right= Null**

[End if]

Step 2: If **Root= Null** Then Set **Root= New**

Exit

[End If]

Step 3: If **Item >= Root → Info** Then

Set **Pointer=Root→ Right**

Set **PionterP= Root**

Else

Algorithm to insert a given element(conti.)

Set **Pointer=Root→ Left**
Set **PionterP= Root**
[End If]
Step 4: Repeat step 5 while Pointer !=Null
Step 5: If **Item >=Pointer→ Info Then**
Set **PionterP=Pointer**
Set **Pointer=Pointer→ Right**
Else
Set **PionterP=Pointer**
Set **Pointer=Pointer→ Left**
[End If]
[End Loop]

Algorithm to insert a given element(conti.)

Step 6: If **Item**< PointerP→Info Then

Set **PointerP→Left**=New

Else

Set **PointerP→Right**= New

[End If]

Step 7: Exit

Complexity of Insertion Process

- Complexity of Insertion process in a Binary search tree is

  O(h), where h is the height of BST.

- If BST is complete binary tree or almost complete binary tree , then the complexity of the insertion process is **O(log2n**)

Deletion of a node from binary search tree

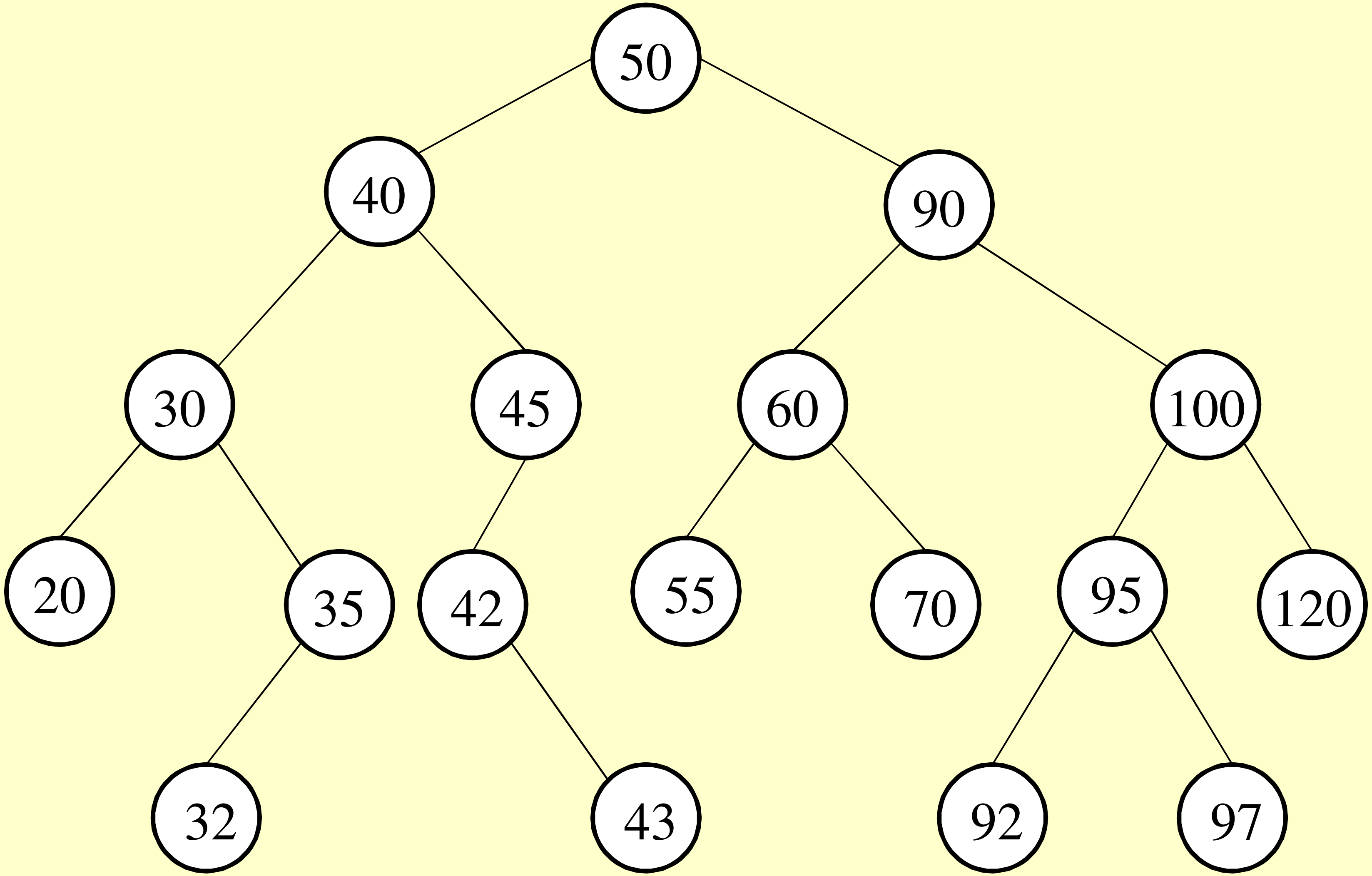
The process of Deletion of a node from BST is a little bit complex than searching and insertion.

The first step for the deletion of a given item from a binary search tree is to locate the node containing the item to be removed and its parent node.

The node to be deleted from the tree may be a leaf node or it may have one child or two children.

Deletion of a node from binary search tree

For example consider a binary search tree shown below:



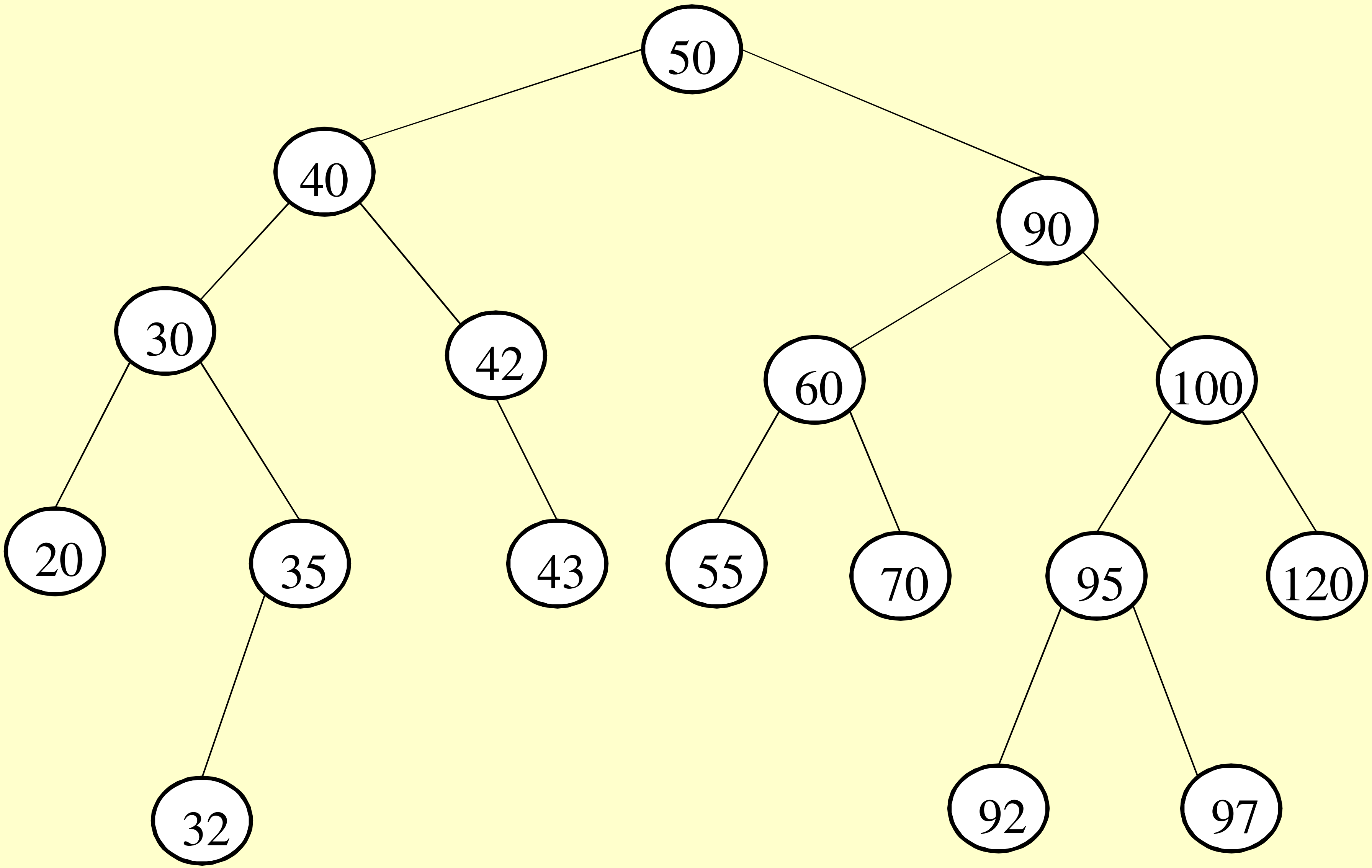
Deletion of a node from binary search tree

Here , in the binary search tree shown, it is very simple to delete the leaf nodes **20, 32, 43, 55, 70, 92, 97** and **120,** because the only thing which is required to be done to change the respective pointer in their parent node to **Null.**

On the other hand when the node to be deleted has only one child, for example the items 35, 42, and 45 in the binary search tree shown above have only one child, the deletion operation is still simple as the node to be deleted will be replaced by its only **child node**.

For example if we want to delete the item 45 from the tree then the node containing 45 will be replaced by its child node and the tree after deletion will become as Shown:

Deletion of a node from binary search tree



Algorithm to delete a given element

DeleteItem(Root, Item)

Step1: Call BSTSearch (Root, Item, Position, Parent)

Step2: If **Position=Null** Then

Print:"Item not found in the tree"

Exit

[End if]

Step 3: If **Position →Left != Null** And **Position → Right !=Null** Then

Call **Delete2(Root, Position, Parent)**

Else

Call **Delete1(Root, Position ,Parent)**

[End If]

Algorithm to delete a given element

Step 4: Deallocate memory held by node **Position**
(Set **Position→Right = Free** And **Free=Position**)
Step 5: Exit

BSTSearch() algorithm has already been explained
Refer this sub algorithm from there.

Algorithm to delete a given element

The below Sub-algorithm delete a node having zero or one child from the binary search tree.

Delete1(**Root, Position, Parent**)

Step1: If **Position→Left=Null** And **Position→Right= Null**

Then

Set **Temp= Null**

Else If **Position →Right!=Null** Then

Set **Temp = Position→Right**

Else

Set **Temp =Position→Left**

[End If]

Step 2: If **Parent= Null** Then

Set **Root= Temp**

Else If **Position= Parent→Left** Then

Algorithm to delete a given element

Set **Parent→Left=Temp**

Else

Set **Parent→Right=Temp**

[End If]

Step 3: Return

Algorithm to delete a given element

The below sub-algorithm delete a node having two children from the binary search tree.

Delete2(**Root, Position, Parent**)

Step1: Set **Pointer = Position →Right And PointerP=Position**

Step2 : Repeat while **Pointer →Left !=Null**

Set **PointerP=Pointer And Pointer= Pointer→Left**

[End Loop]

Step3: Set **Successor =Pointer And PSuccessor=PointerP**

Step4: Call **Delete(Root,Successor,PSuccessor)**

Step5: If **Parent != Null** Then

If **Position =Parent→Left**Then

Set **Parent→Left =Successor**

Else

Algorithm to delete a given element

Set **Parent→Right =Successor** [End If]

Else

Set **Root=Successor**

[End If]

Step6: Set **Successor→Left=Position→Left**

Step7: Set **Successor→Right =Position→Right**

Step8: Return

Finding the smallest element in BST

As in binary search tree every left node is smaller than right node in each subtree of BST.

Therefore to find **the smallest** element in BST we will have to traverse the **left most node** of the BST .

Algorithm to find the smallest element in BST

Step1: If **Root=Null** Then
Print " Tree is Empty"
Exit
Else
Set **Pointer= Root**
[end if]
Step2: Repeat while **Pointer → Left= Null**
Set **Pointer= Pointer→ Left**
[End Loop]
Step3: Set **Min=Pointer→Info**
Step4: Print : **Min**
Step5: Exit

Complexity to find the smallest element in BST

The complexity of finding the smallest element is dependent upon the height of the binary search tree.
So, if the height of the left leg of the tree is highest then the worst case complexity will be **O(h).**
In case the binary search tree is complete or almost complete binary search tree with n elements, the complexity of finding the smallest element will be **O(log2n)**

Finding the largest element in BST

As in binary search tree every right node is smaller than left node in each subtree of BST.
Therefore to find the **largest** element in BST we will have to traverse the **right most node** of the BST .

Algorithm to find the largest element in BST

Step1: If **Root=Null** Then
Print " Tree is Empty"
Exit
Else
Set **Pointer= Root**
[end if]
Step2: Repeat while **Pointer → Right= Null**
Set **Pointer= Pointer→ Right**
[End Loop]
Step3: Set **Max=Pointer→Info**
Step4: Print : **Max**
Step5: Exit

Complexity to find the largest element in BST

The complexity of finding the largest element is dependent upon the height of the binary search tree.
So, if the height of the right leg of the tree is highest then the worst case complexity will be **O(h).**
In case the binary search tree is complete or almost complete binary search tree with n elements, the complexity of finding the largest element will be **O(log2n)**