Binary Search Tree
Dec. 11(L 20)

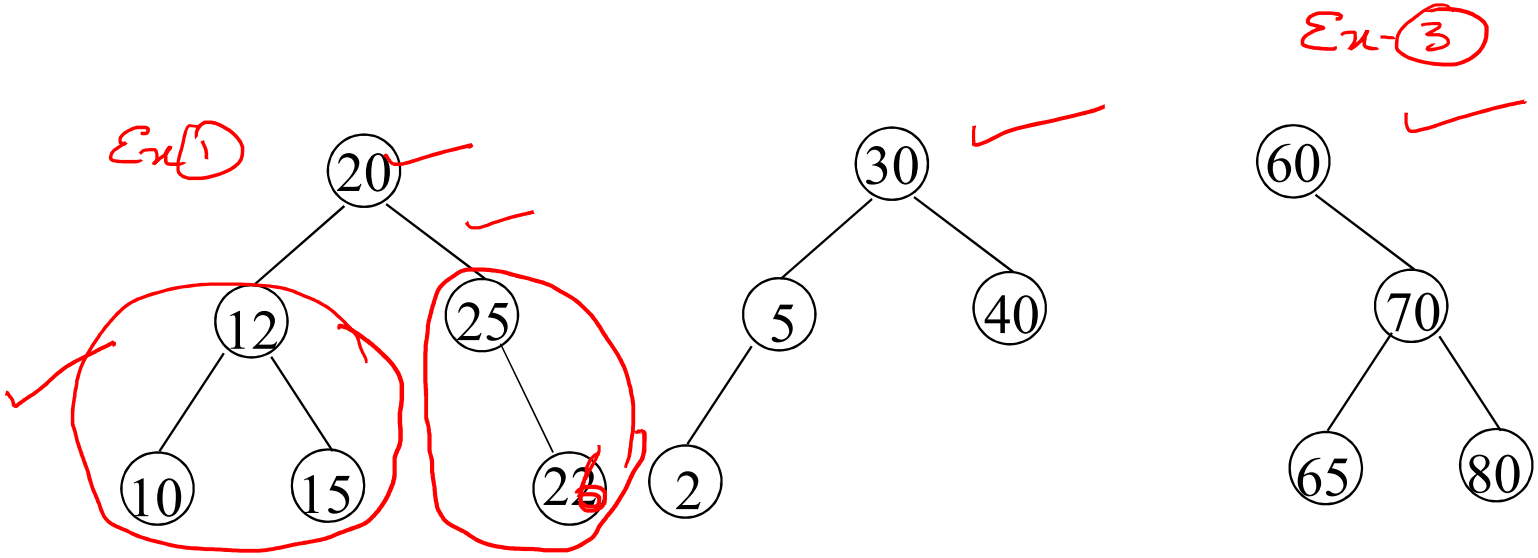
Binary Search Tree (BST)

Binary search tree

- BST is a binary tree which may be empty. If exists the it satisfies the following properties

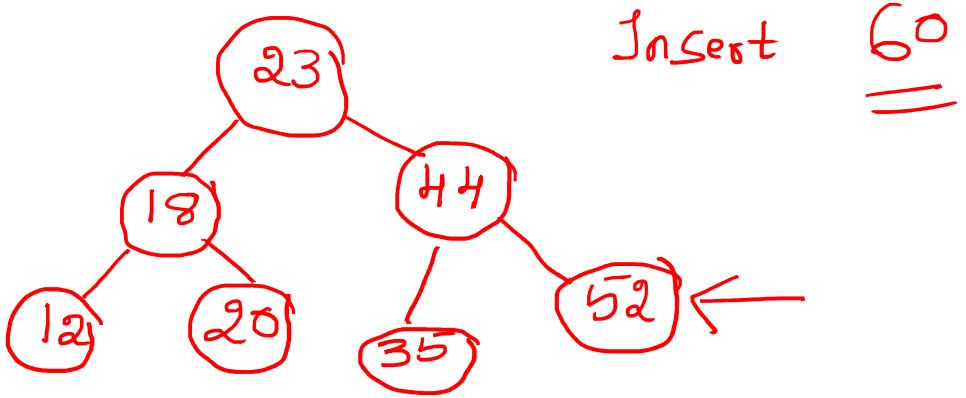
  - Every element has a unique key.
  - The keys in a nonempty left subtree (right subtree) are smaller (larger) than the key in the root of subtree.
  - The left and right subtrees are also binary search trees.

Examples of Binary Search Trees

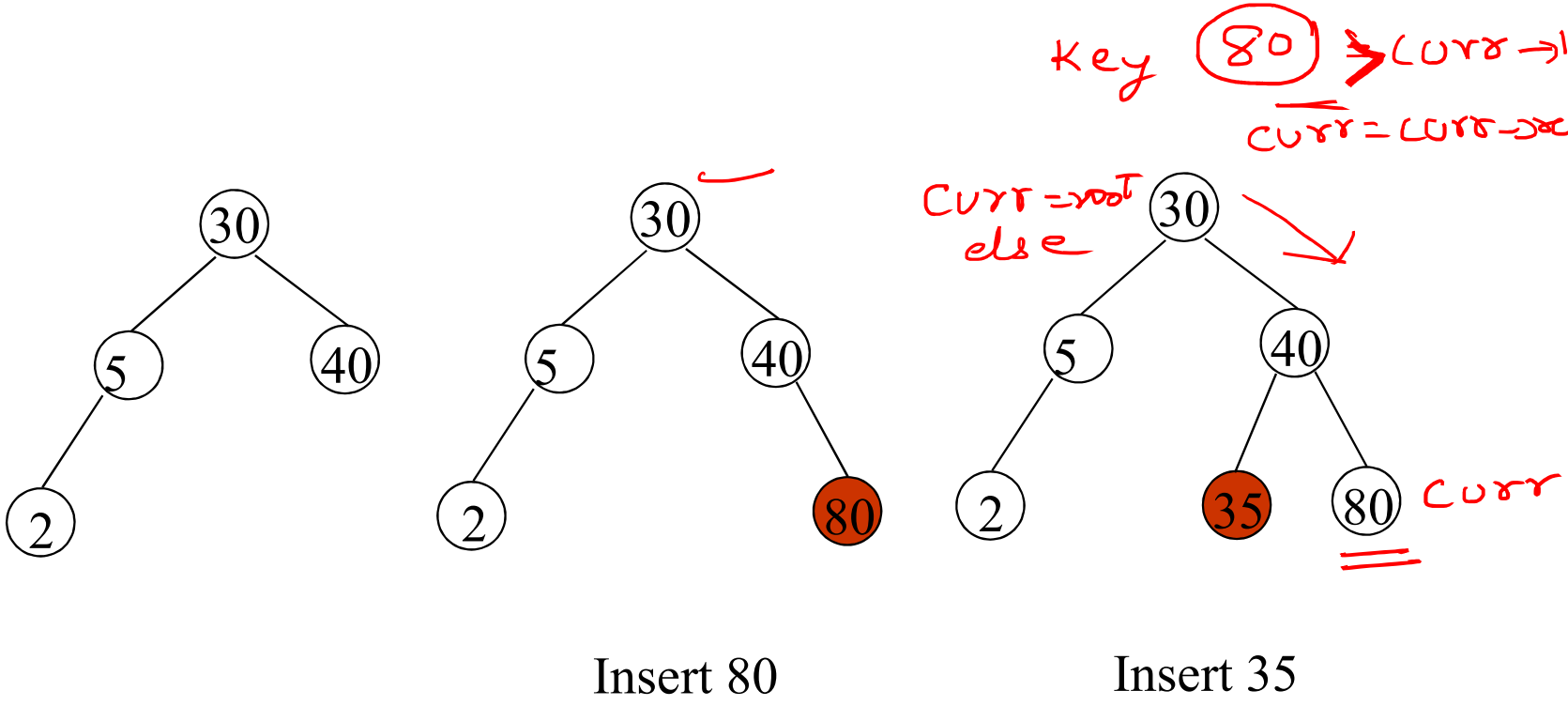


Create a Binary Search Tree

Key values are input in the following order: 23, 44,18,20,52,12,35



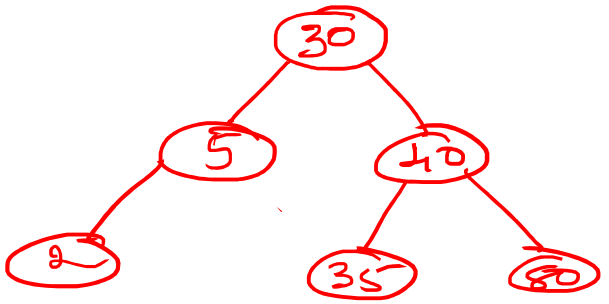
Insert Node in Binary Search Tree

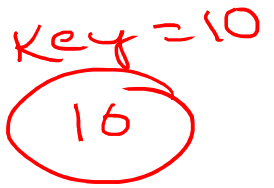


Searching an element in BST

// Returns a pointer to the node that contains search element else // returns NULL

```
bstree search(bstree *tree, int key)
{
 while (tree) {
   if (key == tree->data) return tree;
   if (key < tree->data)
      tree = tree->left_child;
   else tree = tree->right_child;
  }
  return NULL;
}
```





Searching an element in BST-Recursive

```
bstree search(bstree *root,
                      int key)
{
/* return a pointer to the node that contains key.
  If there is no such
  node, return NULL */
                (root == NULL)
   if (!root) return NULL;
   if (key == root->data) return root;
   if (key < root->data)
        return search(root->left_child,
                      key);
   return search(root->right_child,key);
}
```

Inserting an element in BST

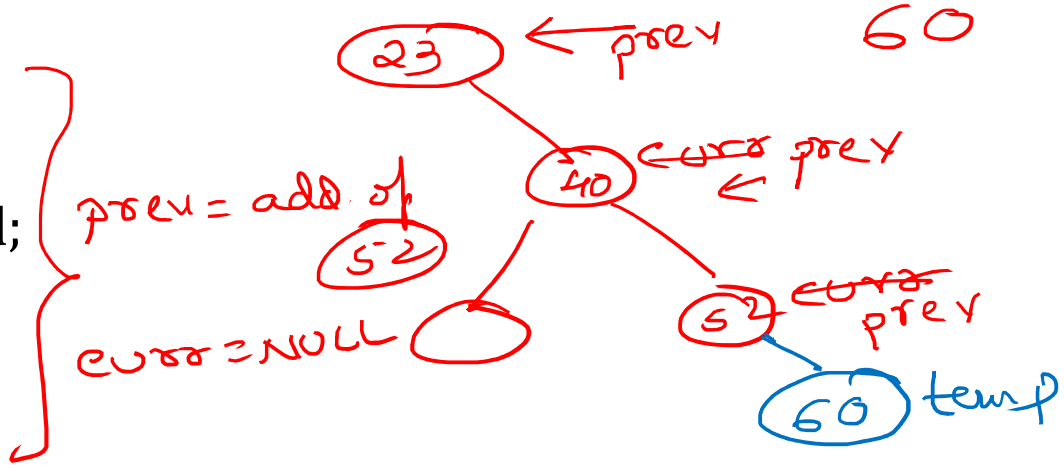
```
void bstree:: create()
{ bstree *curr=root, *prev=NULL;
  int ele;
  cout<<"enter the key element to be inserted\n"; cin>>ele;
  bstree *temp=new bstree(ele,NULL,NULL);
 if (root==NULL)
        root=temp;
else
```

loop to run n times, n → no. of nodes

Contd. In next slide

Inserting an element in BST

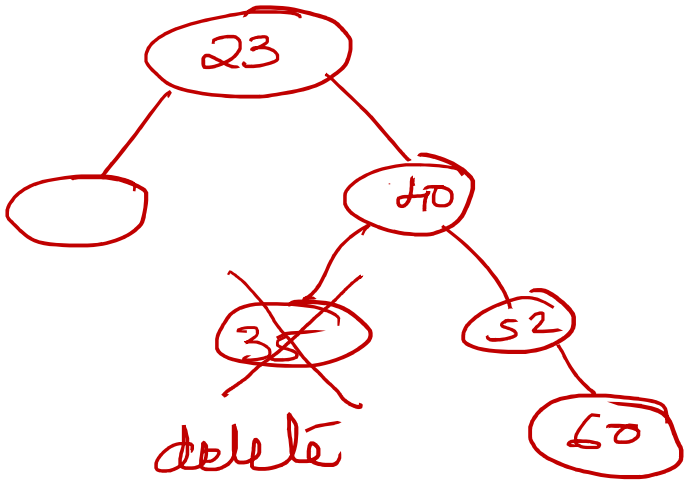
```
{
 while (curr)
{ prev=curr;
  if(ele>curr->data)  curr=curr->rchild;
  else
                curr=curr->lchild;
}

If (curr)
        if ( ele > prev->data)
                prev->rchild=temp;
        else
                prev->lchild =temp;
}
}
```



Deletion for A Binary Search Tree

The node to be deleted may be a:

- ✓ Leaf node
- ✓ Node with degree one
- ✓ Node with degree two



Consider the following pointer variables of bstree type:

curr : node to be deleted
prev : Parent of curr
temp : Node to be placed in the place of curr

Deletion for A Binary Search Tree

Case I: Deleting a Leaf node

curr, prev

Write a loop to get the add of the node to be deleted (curr) and address of its parent (prev)

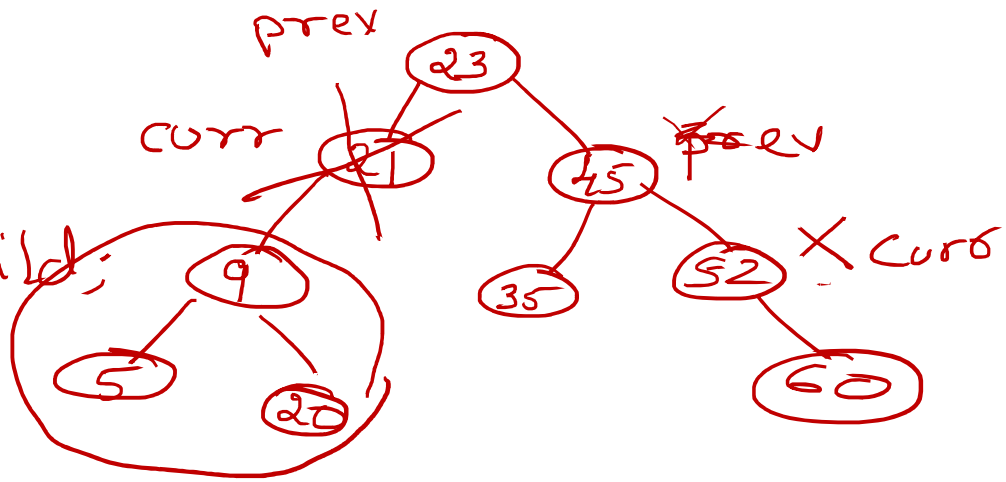
```
if (curr == prev->lchild)
        prev->lchild = NULL
else
        prev->rchild = NULL
```

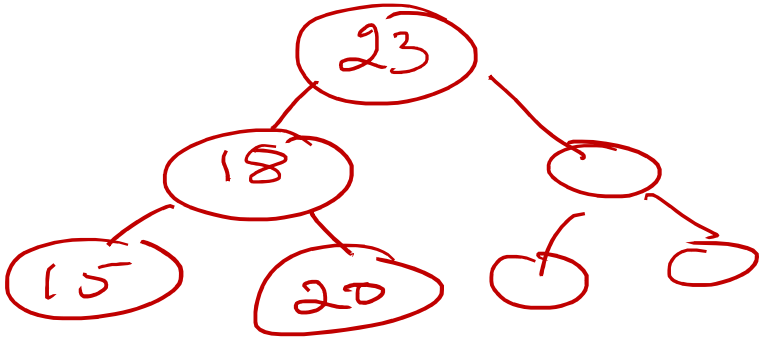
Deletion for A Binary Search Tree

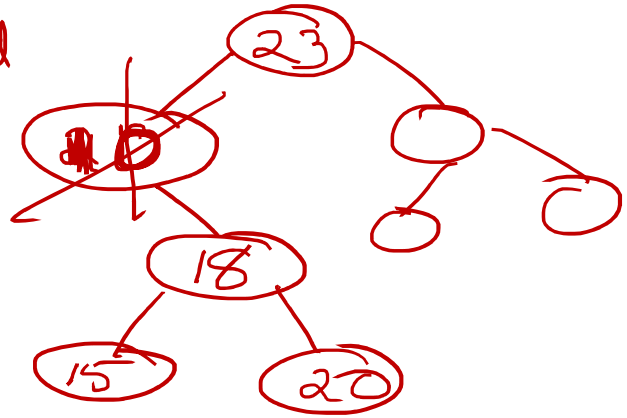
if (curr == prev->lchild)

Case II: Deleting a node with degree 1

```
if(curr->rchild == NULL)
    prev->lchild = curr->lchild;
else
if(curr->lchild == NULL)
    prev->lchild = curr->rchild
```

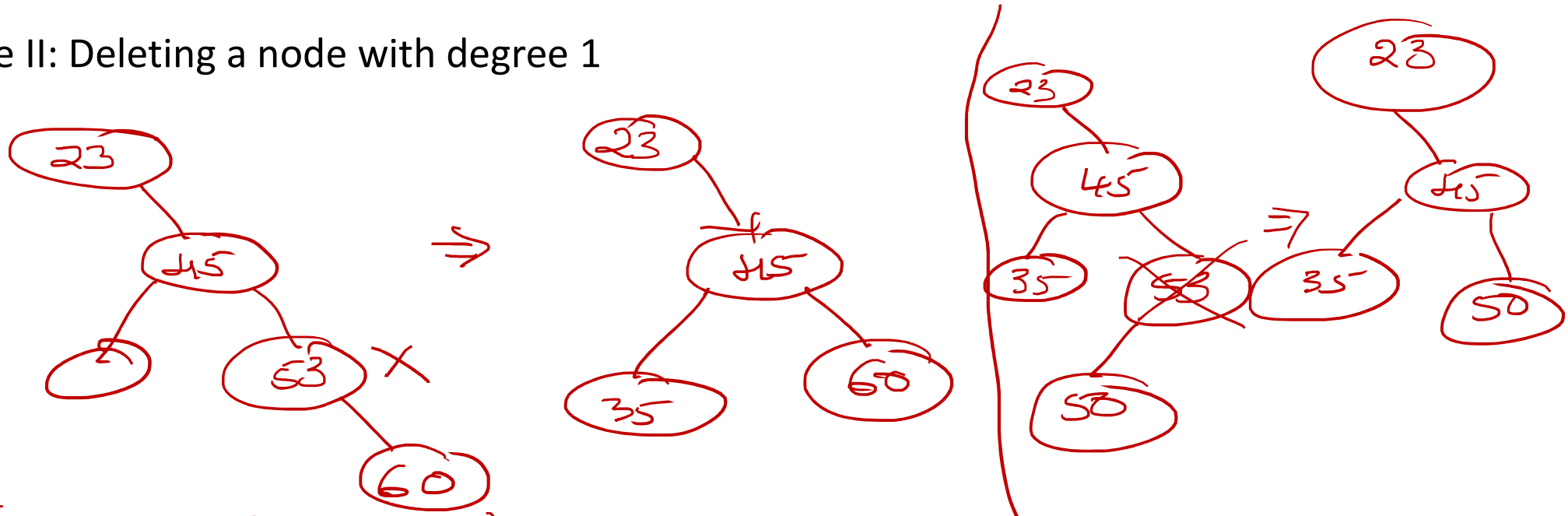






Deletion for A Binary Search Tree

Case II: Deleting a node with degree 1



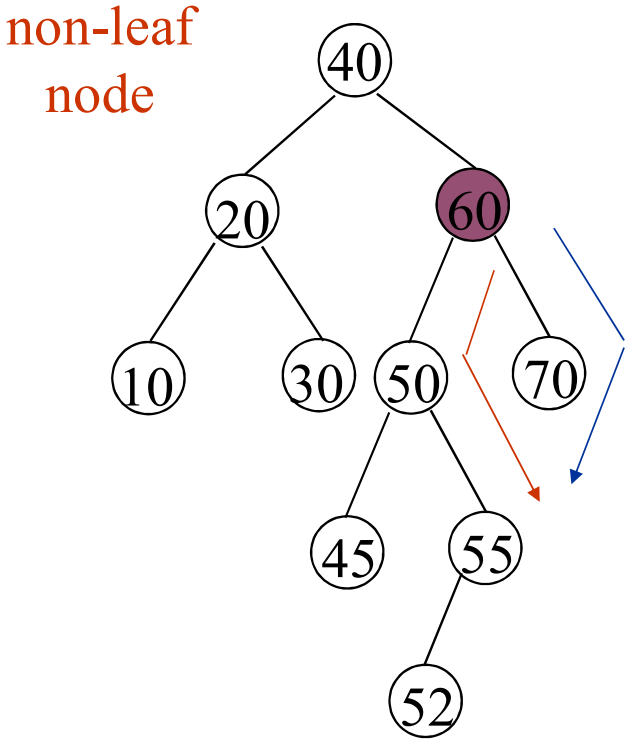

```
if(curr=prev->rchild)
  { if(curr->lchild=NULL)
        prev->rchild = curr->rchild;
  } else  prev->lchild = curr->lchild;
```

Deletion for A Binary Search Tree

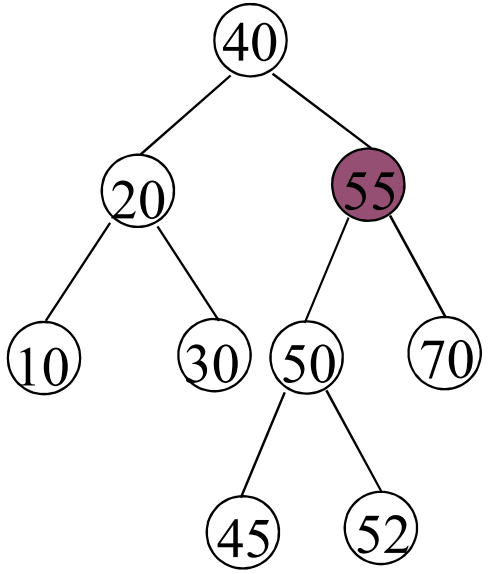
Case I: Deleting a node with degree 1

Deletion for A Binary Search Tree

Case IIII: Deleting a node with degree 2



Before deleting 60

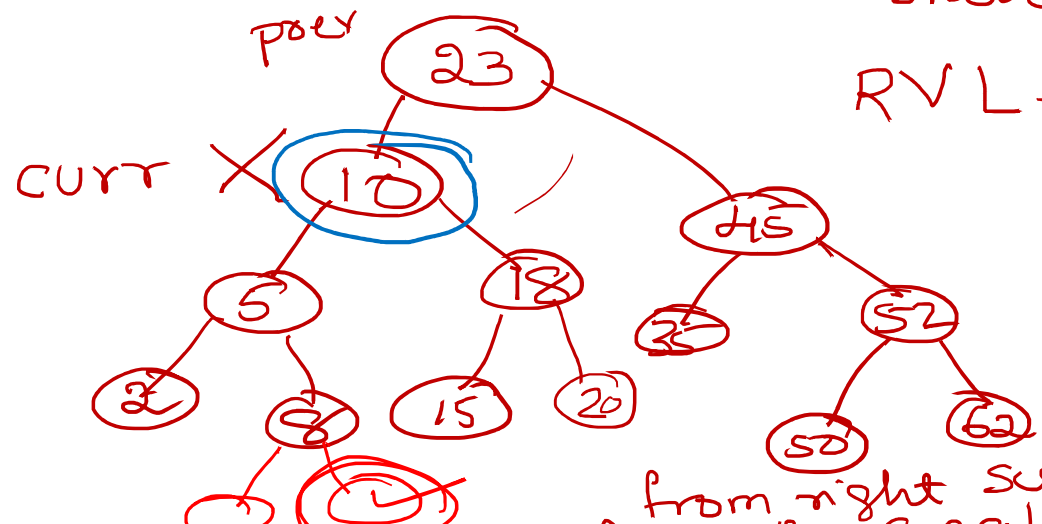


After deleting 60

Deletion for A Binary Search Tree: Function

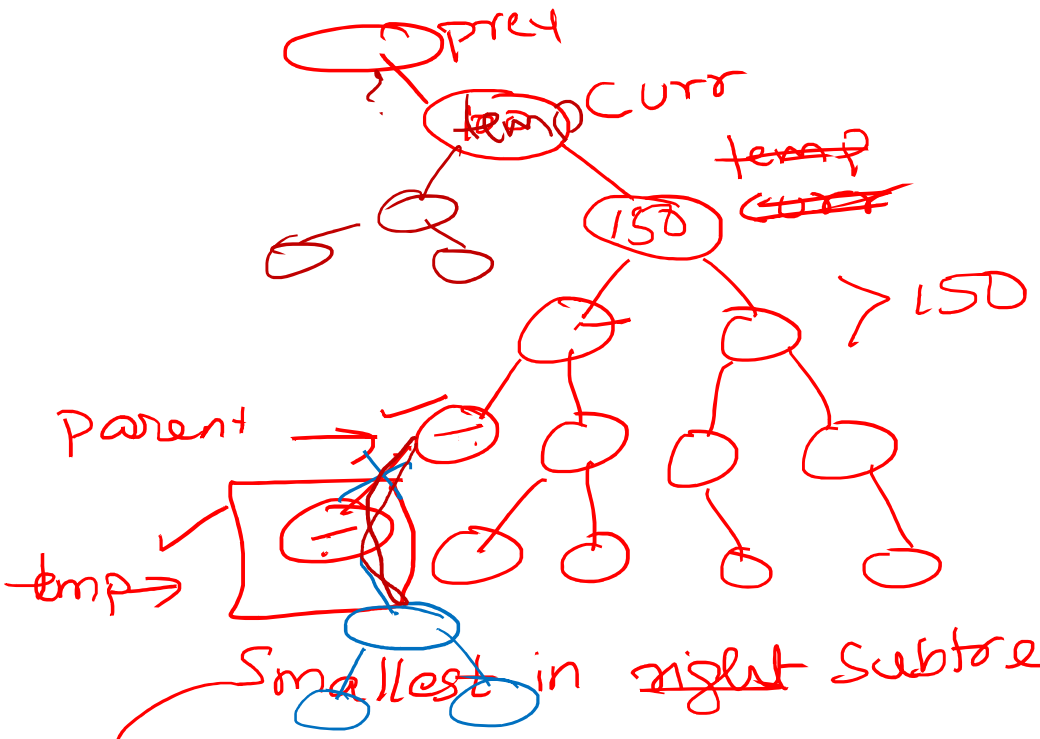
Inorder: 2 5 8 10 15 18 20 23 35 45 50 52 62

RVL ⇒ Descending



① Replace by a node with smallest key value ⇒ 15 from right subtree ✓

② Replace by a node from the left subtree ~~or~~ node with ~~highest~~ largest key value ⇒ 8




```
temp = curr->rchild
while (temp->lchild != NULL)
{ parent = temp;
  temp = temp->lchild;
}
```
(1)

⇒ left most node

⇒ right most node

```
parent->lchild = temp->rchild;
temp->lchild = curr->lchild;
temp->rchild = curr->rchild;
```
(2)

(3)
```
if (prev->lchild == curr)
    prev->lchild = temp;
else
    prev->rchild = temp;
```

Creati a BST

100, 98, 213, 84, 76, 36, 123, 146, 190

8, 10, 14