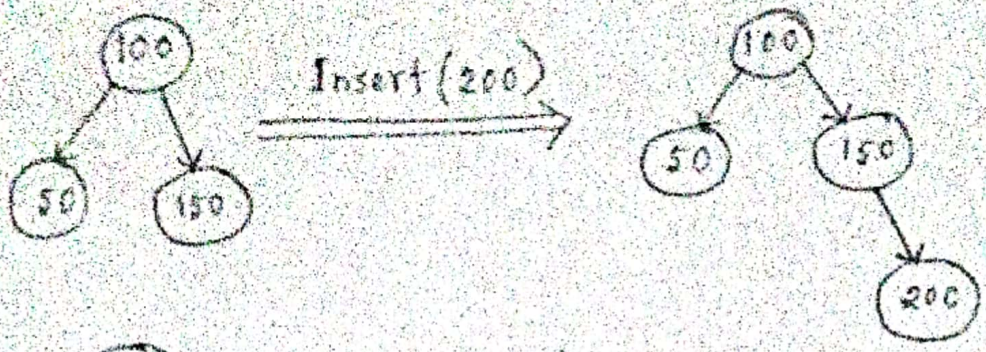
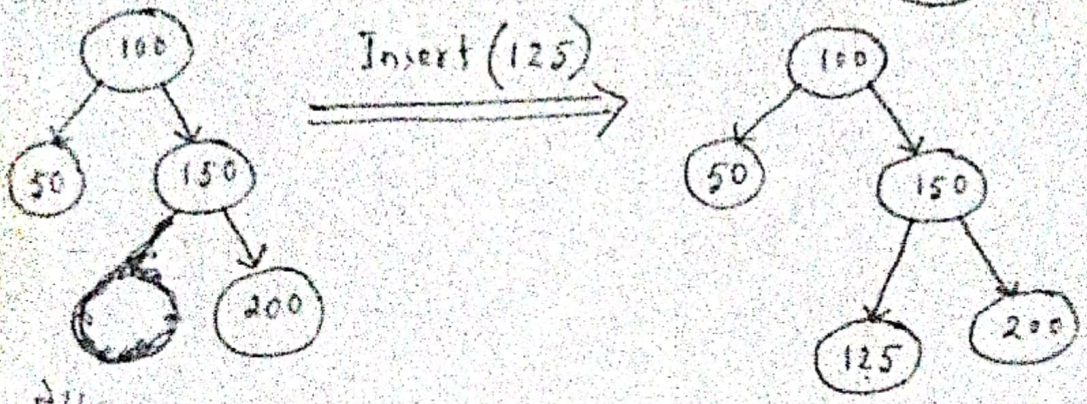
Inserting a node in a binary search tree

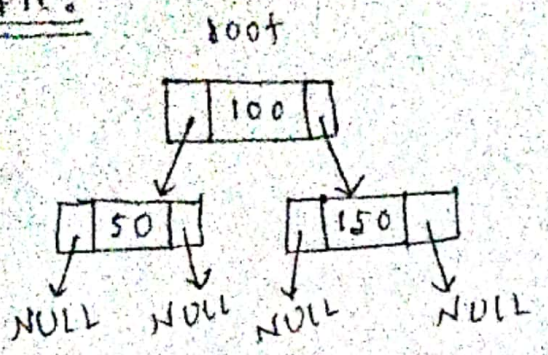


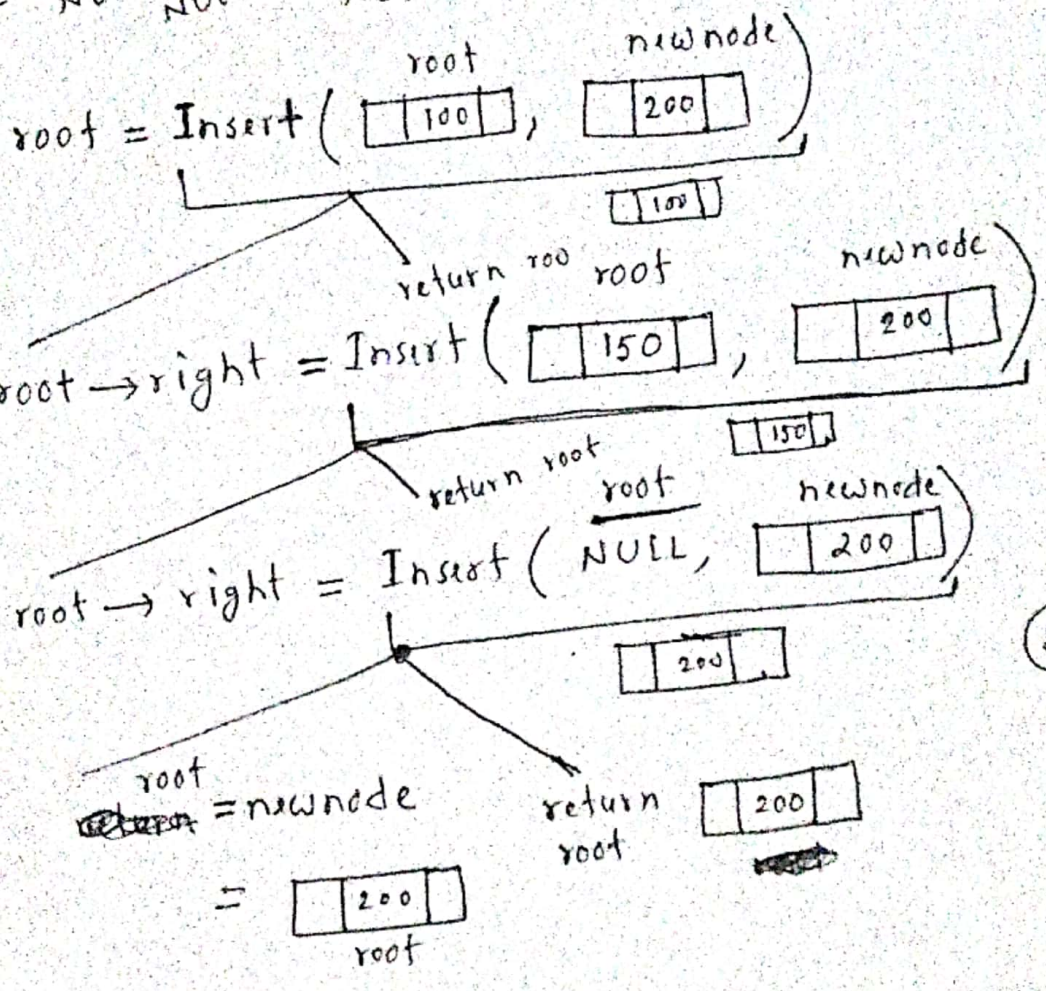


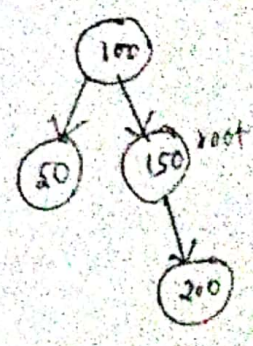
Recursive Algorithm

```
node *Insert( node *root, node *newnode){
    if (root == NULL)
    root= newnode;
    else if (root->data < newnode->data)
        root->right = Insert (root->right, newnode);
    else if (root->data > newnode->data)
        root->left = Insert (root->left, newnode);
    return root;
}
```

iv Example:







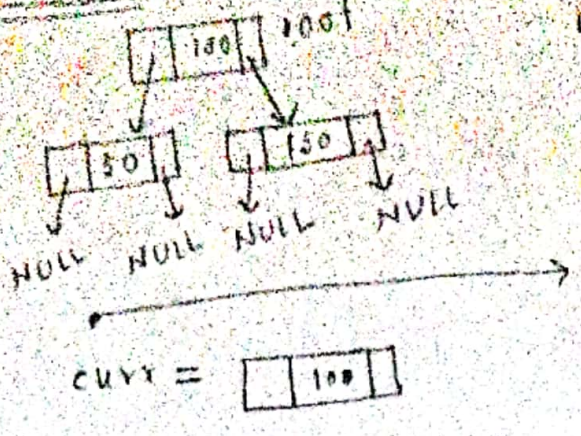
Iterative Algorithm


```
node *tinsert(node *root, int key){
    node *toinsert, *curr, *prev;

    toinsert = (node*)malloc(sizeof(node));
    toinsert->data = key;
    toinsert->left = NULL;
    toinsert->right = NULL;

    curr = root;
    prev = NULL;
    while (curr != NULL) {
        prev = curr;
        if (curr->data > key)
            curr = curr->left;
        if (curr->data < key)
            curr = curr->right;
    }
    if (prev == NULL)
        root = toinsert;
    else if (prev->data > key)
        prev->left = toinsert;
    else if (prev->data < key)
        prev->right = toinsert;
    return root;
}
```

Example:



prev = NULL

while (True) ⇒ prev = 100 if (100 > 200) False if (100 < 200) True curr = 150

while (True) ⇒ prev = 150 if (150 > 200) False if (150 < 200) True curr = NULL

while (False)

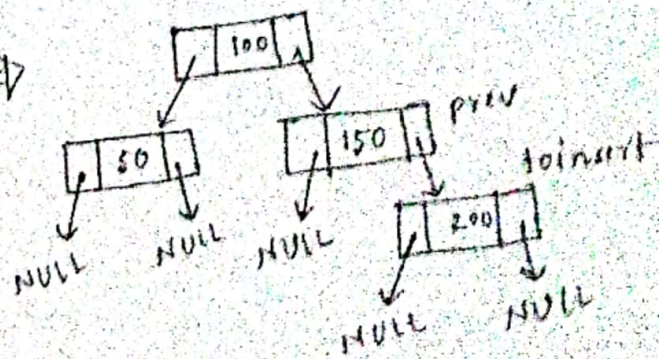
if (False)

else if (150 > 200) False

else if (150 < 200) True

prev→right = toinsert ⇒

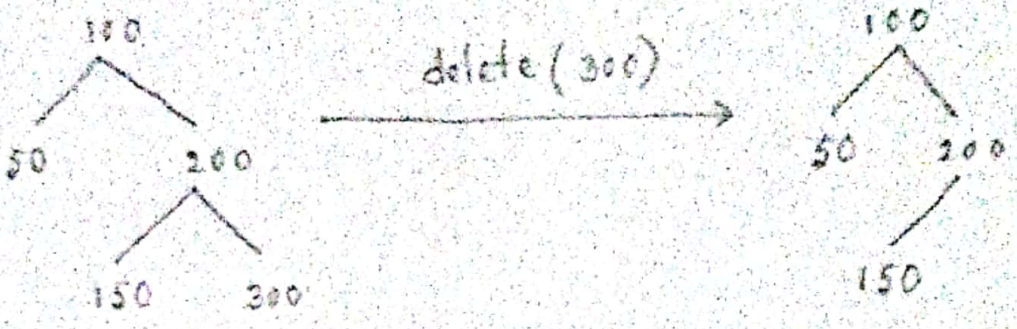
= 200 (toinsert)



Deletion from Binary Search Tree

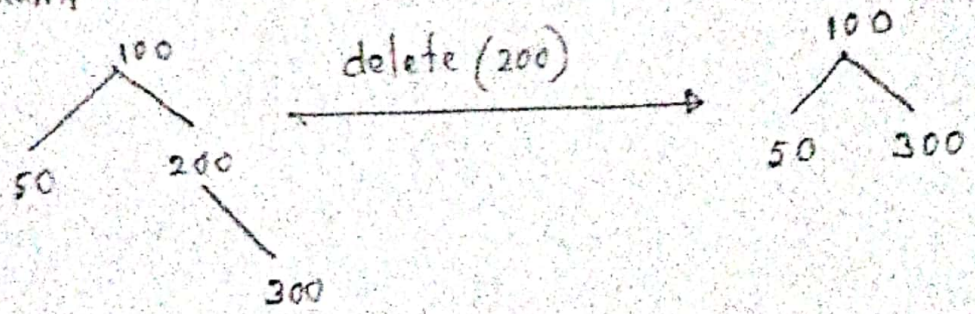
1. Leaf node:
   If the node is leaf (both left and right will be NULL), remove the node directy and free its memory

   Example:



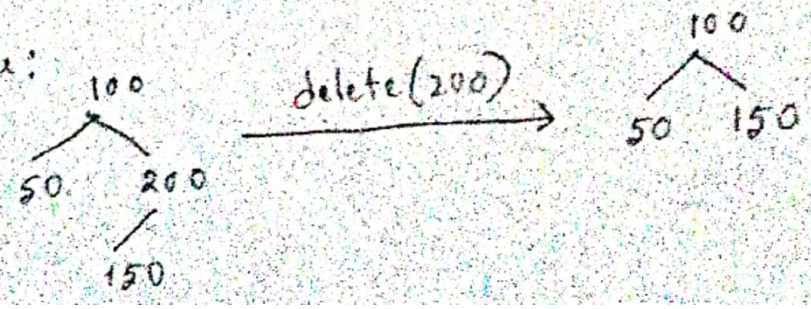
2. Node with Right child:
   If the node has only right child (left will be NULL), make the node points to the right node and free the node.

   Example:



3. Node with Left child
   If the node has only left child (right will be NULL), make the node points to the left node and free the node.

   Example:

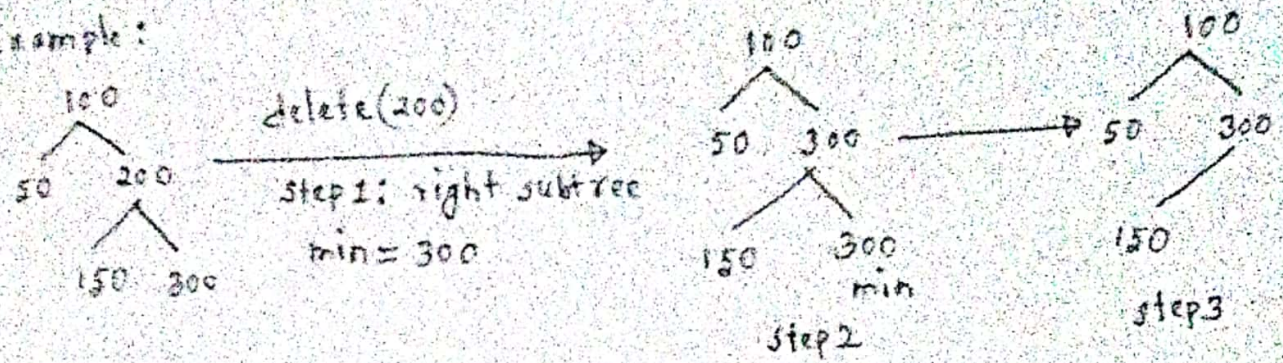


4. Node has both left and right child

If the node has left and right child

1. Find the smallest node in the right subtree, say min
2. Make node→data = min
3. Again delete the min node

Example:



Deletion function

```
node *Delete(node *root, int val){
    if (root == NULL) return NULL;
    if (root->key < val)
        root->right = Delete(root->right, val);
    else if (root->key > val)
        root->left = Delete(root->left, val);
    else{
        if(root->left == NULL && root->right == NULL){
            free(root);
            return NULL;
        }
        else if(root->left == NULL){
            node *temp = root->right; free(root); return temp;
        }
        else if(root->right == NULL){
            node *temp = root->left; free(root); return temp;
        }
        else{ int rightMin = getRightMin(root->right);
            root->key = rightMin;
            root->right = Delete(root->right, rightMin);
        }
    return root;
}
```

```
int getRightMin(node *root){
    node *temp = root;
    while(temp->left != NULL)
        temp = temp->left;
    return temp->key;
}
```