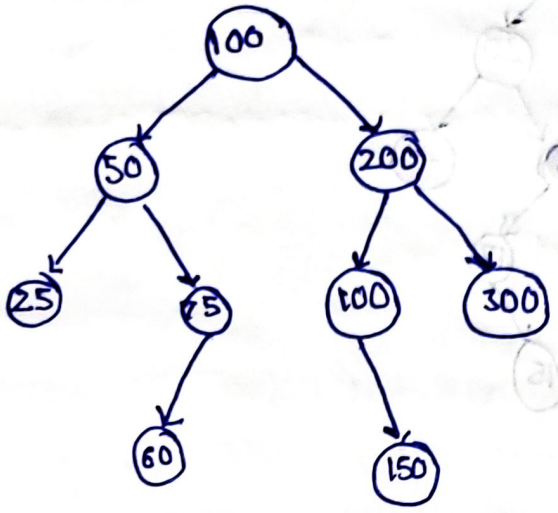
14.05.23

## Binary Search Tree :-

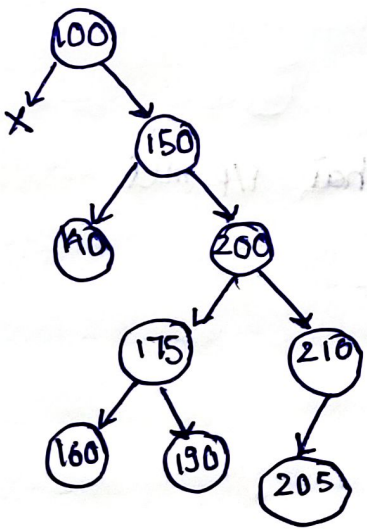



BST: हर Node k लिए उसके left subtree में जो data पड़ा हुआ वो chota होगा root से and right subtree में जो data पड़ा है वो root से बड़ा होगा.

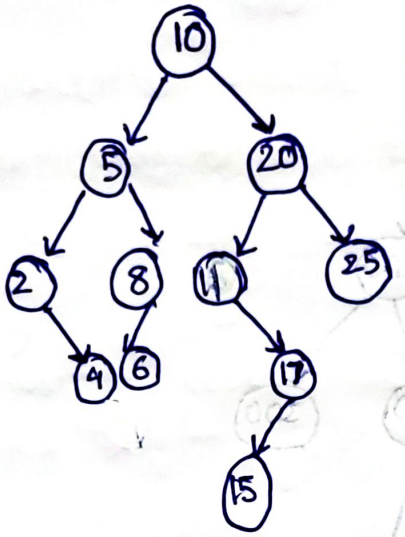
- We are not dealing with duplicate values in BST, अगर होगा तो ques में specifically mentioned होगा how to deal.

i/p- 100 150 200 175 160 140 210 205 190

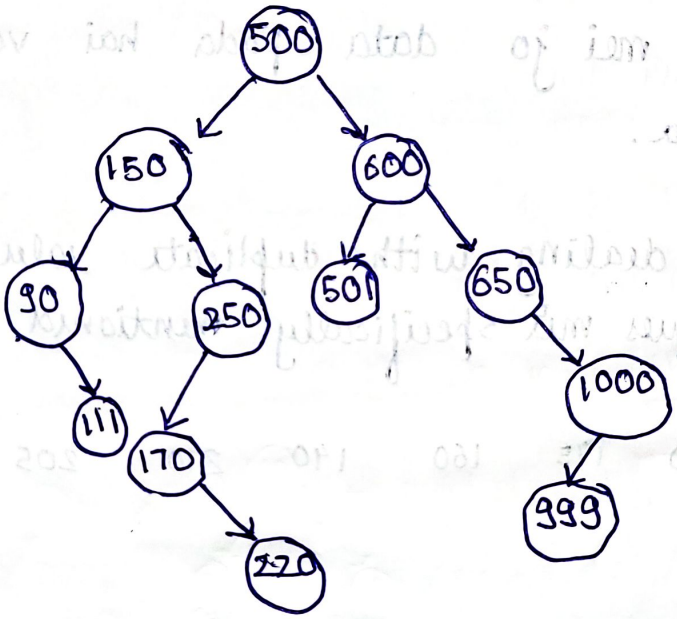
root=NULL



i/p - 10, 20, 5, 11, 17, 2, 4, 8, 6, 25, 15




i/p - 500 150 250 600 650 170 90 220
501 1000 111 999




Note → -1 aane pe rukna hai i/p mei

```
class Node {
        public:
        int data;
        Node * left;
        Node * right;
        Node(int data){
                this -> data = data;
                this -> left = NULL;
                this -> right = NULL;
        }
};
main(){

   Node * root = NULL;
   // Enter data for Node
   takeInput();

   return 0;

}

void takeInput(Node*& root){
      cin >> data;
      while(data != -1){
            root = insertIntoBST(data);
            cin>>data;
      }
}

Node * insertIntoBST(Node* root, int data) {
        if(root == NULL)
              // creating first node
              root = new Node(data);
              return root;
        }
```

```
if(root -> data > data){
        root -> left = insertIntoBST (root -> left, data);
}else{
    //insert into right
    root -> right = insertIntoBST (root -> right, data);
}
return root;
}
```

Inorder

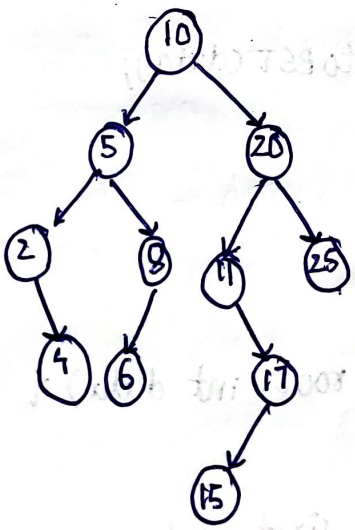
2 4 5 6 8 10 11 15 17 20 25 (LNR)

Preorder

10 5 2 4 8 6 20 11 17 15 25 (NLR)

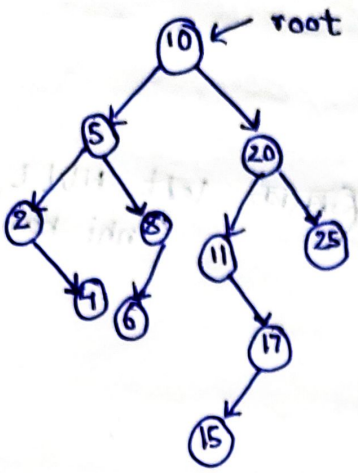
Postorder

4 2 6 8 5 15 17 11 25 20 10 (LRN)



Note- Inorder of BST gives sorted order.

- Searching:




target = 17

T.C.

avg case(BST) → height (h) → O(log n)

```
// For only unique values
bool findNodeInBST(Node* root, int target){
    //base case
    if(root == NULL)
        return false;

    if(root->data == target)
        return true;

    if(target > root->data){
        return findNodeInBST(root->right, target);
    }
    else{
        return findNodeInBST(root->left, target);
    }
}
```

Q. Minimum Value in BST

```
int minVal(Node* root){
    Node* temp = root;
    if(temp == NULL){
        return -1;
    }
    while(temp -> left != NULL)
        temp = temp -> left;
    }
    return temp -> data;
}
```

(जबतक left NULL nhi hai!)

Q. Max^m value in BST.

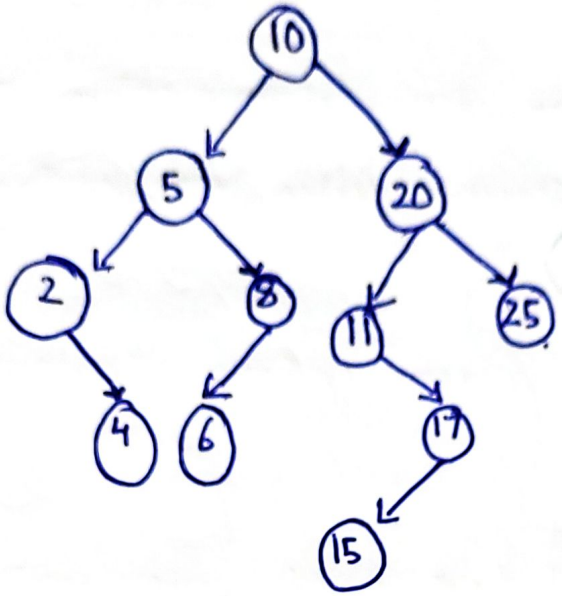
```
int maxVal(Node* root){
    Node* temp = root;
    if(temp==NULL){
        return -1;
    }
    while(temp -> right != NULL)
        temp = temp -> right;
    }
    return temp -> data;
}
```

(जबतक Right NULL nhi hai!)

↑ left ST ← max

- Inorder predecessor / successor → right ST + min

IP → left subtree ka max^m value



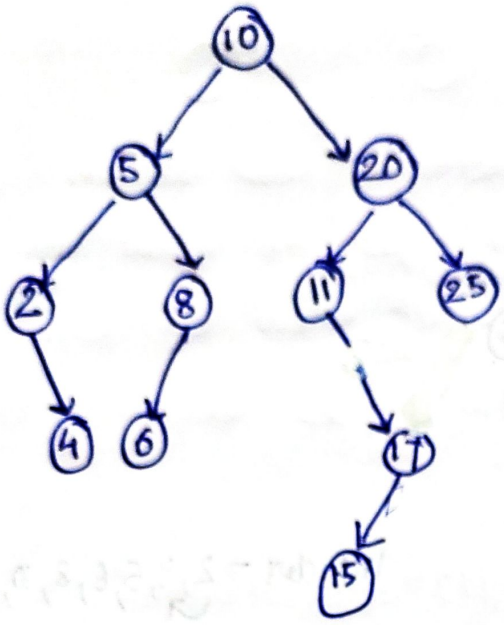

Inorder - 2, 4, 5, 6, 8, 10, 11, 15, 17, 20, 25

Predecessor - node just before the node
i.e. predecessor of 2 is 5

Successor - node just after the node
i.e. successor of 2 is 4

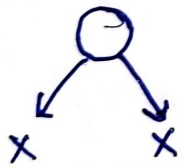
# Deletion in BST :-

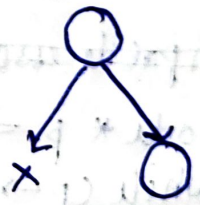
Interview

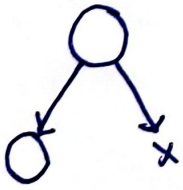


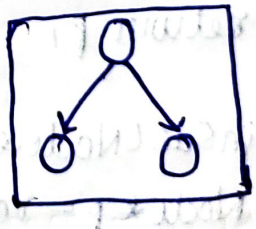

target = 25 तक पहुंचना है.

4 cases :- (For deleting any node)

a)  return NULL

b)  return root→right

c)  return root→left

d) 

## Algorithm: -

- Reach the node (recursively)

```
if (root → data == x){
    ____
    ____
}
else if (root → data > x){
    // left part (R.C)
}
else {
    // Right part ki R.Call
}
```

---

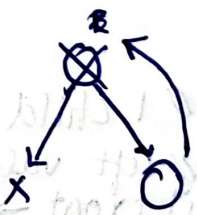
## Node to Delete

↳ 0 child

```
delete node;
return NULL
```

↳ 1 child

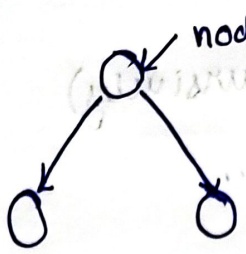
or



↳
```
temp = root → left
delete root;
return temp
```

```
temp = root → right
delete root;
return temp
```

↳ 2 children




① left mai se max^m value ko upar copy krdo
• fir jis node ko copy kiya usko vo jaha pe tha vha se hata do.

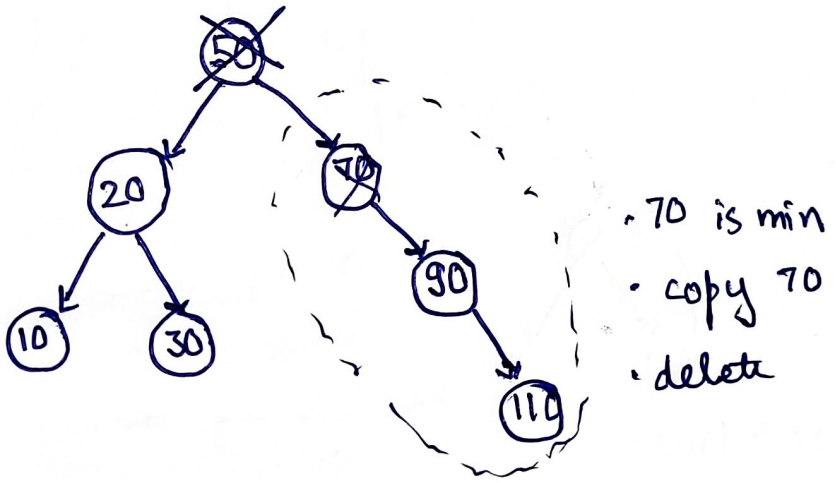
OR

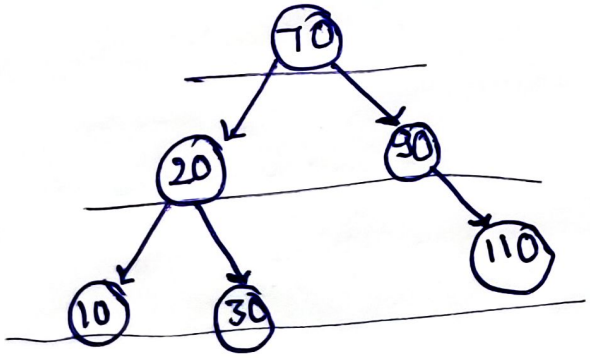
② Right subtree se minimum value ko same.

Code:-

```
Node *  deleteFromBST(Node* root, int val){
    //base case
    if(root==NULL)
        return root;
    if(root->data == val){
        // 0 child
        if(root->left == NULL && root->right == NULL){
            delete root;
            return NULL;
        }
        // 1 child
        // left value
        if(root->left != NULL && root->right == NULL){
            Node* temp = root->left;
            delete root;
            return temp;
        }
        //right child
        if(root->left == NULL && root->right != NULL){
            Node* temp = root->right;
            delete root;
            return temp;
        }
```

```
// 2 child
if(root -> left != NULL && root -> right != NULL){
        int mini = minVal(root -> right) -> data;
        root -> data = mini;
        root -> right = deleteFromBST(root -> right, mini);
        return root;
}
}
```




Average case T.C. of deletion is O(h).