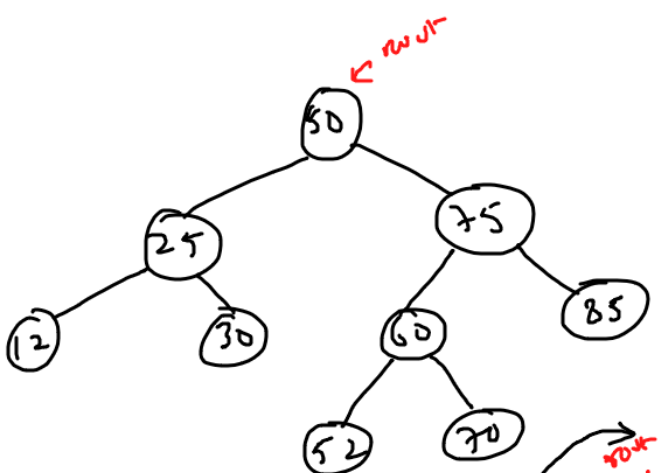
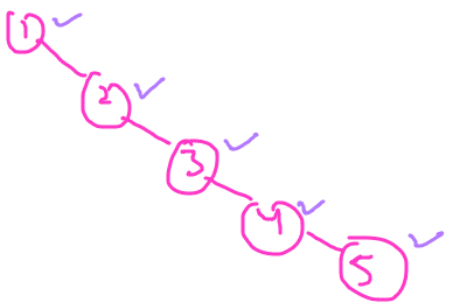
$\log(n)$

(1) search $\leftarrow T(n) = O(n)$





(2) max node

(3) min node

tree root = insert(tree root, 11)

Max Node

```
int findMax(Node root)
{
    while(root.right)
    {
        root = root.right;
    }
    return root->data;
}
```

Min Node

```
int findMin(Node root)
{
    while(root.left)
    {
        root = root.left;
    }
    return root->data;
}
```

Search Node

```
Node search(Node root, int x)
{
    if (root == null || root.data == x)
        return root;
    if( x < root.data)
        return search(root.left, x);
    return search(root.right, x);
}
```

```
Node(int x)
{
    data = x;
    left = right = null;
}
```

↑ constructor

Insertion :-

```
Node create (int x)
{
    Node np = new Node (x);
    return np;
}
```

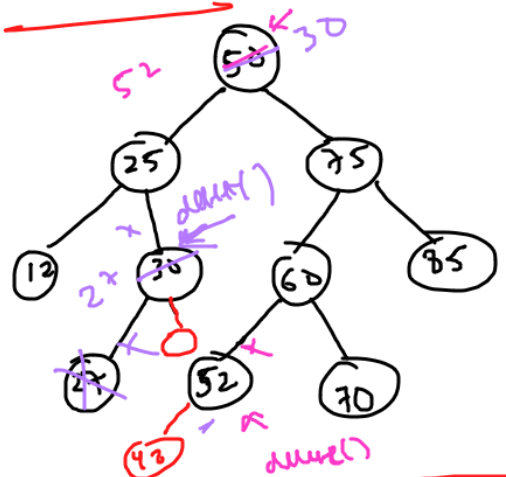
main() :-

~~tree.insert(10)~~

✓ tree.root = tree.insert (root, 20);

```
Node insert (Node root, int x)
{
    if (root == null)
    {
        return new Node (x);
    }
    if (x < root.data)
        ✓ root.left = insert (root.left, x);
    else
        root.right = insert (root.right, x);
    return root;
}
```

Deletion



(1) a leaf node

(2) a node with one child node

(3) a node with both child nodes

inorder successor → can't have left child

inorder predecessor → can't have right child

findMin (root.right) → inorder successor

findMax (root.left) → inorder predecessor