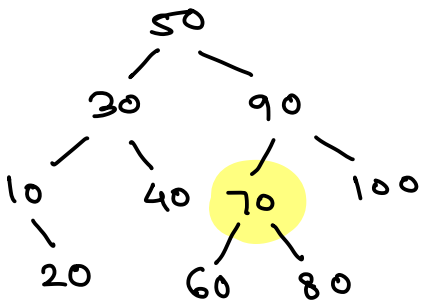
Data Structure & Algorithms

Sunbeam Infotech

Agenda

- BST – add node (non-recursive)
- BST – Binary Search (non-recursive)
- BST – add node (recursive)
- BST – Binary Search (recursive)
- BST – Pre-order traversal (recursive)
- BST – In-order traversal (recursive)
- BST – Post-order traversal (recursive)
- BST – Delete All (recursive)
- BST – Height (recursive)
- BST – Pre-order traversal (non-recursive)
- BST – In-order traversal (non-recursive)
- BST – Post-order traversal (non-recursive)
- BST – BFS & DFS (non-recursive)

BST - Binary Search

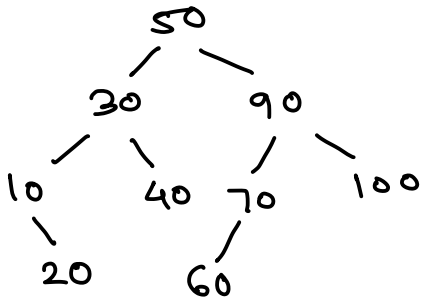


key = 80

key = 65

```
node * binsearch (int key) {
    node * trav = root;
    while (trav != null) {
        if (key == trav->data)
            return trav;
        if (key < trav->data)
            trav = trav->left;
        else
            trav = trav->right;
    }
    return NULL;
}
```

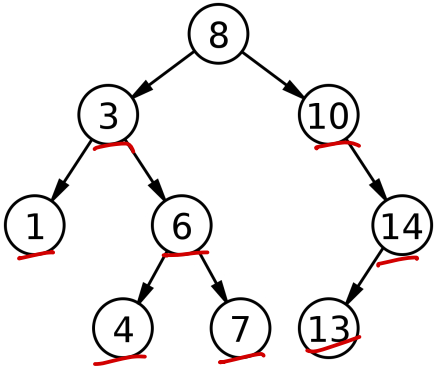
BST - add - recursive



new val = 80

```
void add(int val) {
    if(root == null)
        root = new node(val);
    else
        add(root, val);
}
```

```
void add(node * trav, int val) {
    if(val < trav->data) {
        if(trav->left == NULL) {
            trav->left = new node(val);
            return;
        }
        add(trav->left, val);
    }
    else {
        if(trav->right == NULL) {
            trav->right = new node(val);
            return;
        }
        add(trav->right, val);
    }
}
```

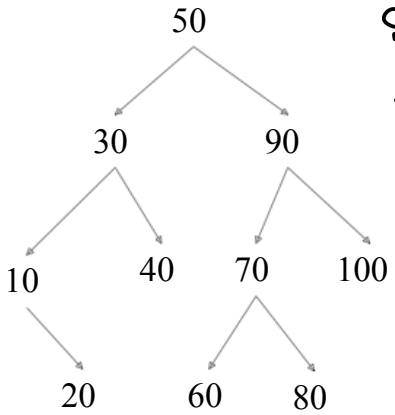


Inorder: 1 3 4 6 7 8 10 13 14

Preorder: 8 3 1 6 4 7 10 14 13

Postorder: 1 4 7 6 3 13 14 10 8

BST – recursive bin search



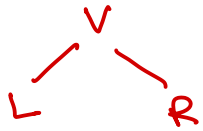
comp key with root;
if same, return ptr;

if key is smaller,
find in left subtree.

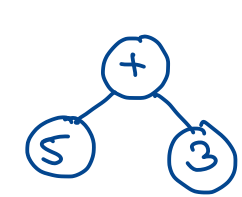
else
find in right subtree.

Key = 80

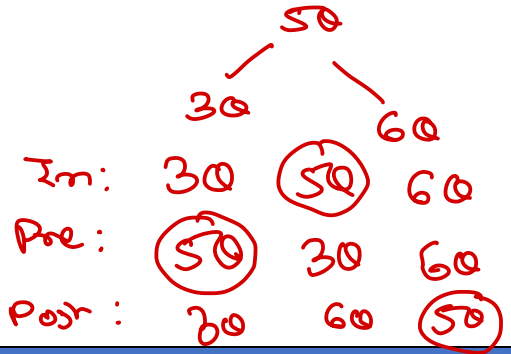
Binary Tree Traversal



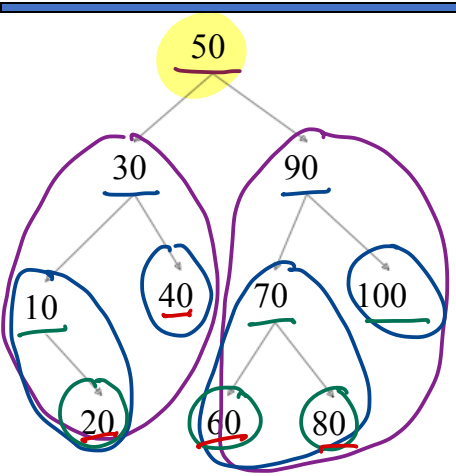
- **In-order** □ L V R
- **Pre-Order** □ V L R
- **Post-Order** □ L R V
- The traversal algorithms can be implemented easily using recursion.
- Non-recursive algorithms for implementing traversal needs stack to store node pointers.



In: 5 + 3
Pre: + 5 3
Post: 5 3 +



BST – Recursive Algorithm – preorder : V L R

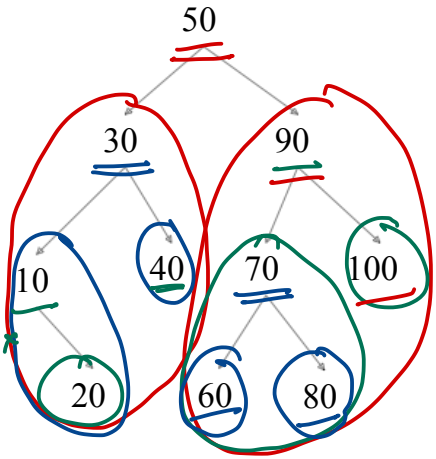


50 30 10 20 40
90 70 60 80 100

```
void preorder(node *trav) {
    if (trav == NULL)
        return;
    cout << trav->data;
    preorder(trav->left);
    preorder(trav->right);
}

void preorder() {
    preorder(root);
}
```

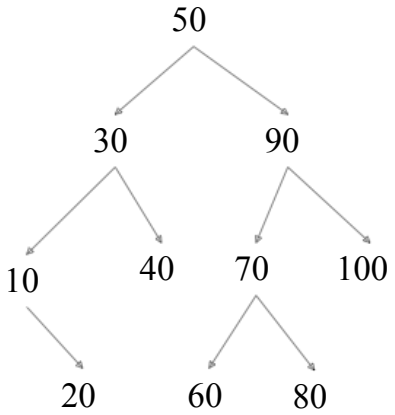
BST – Recursive Algorithm – inorder → L V R



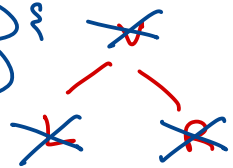
10 20 30 40 50

60 70 80 90 100

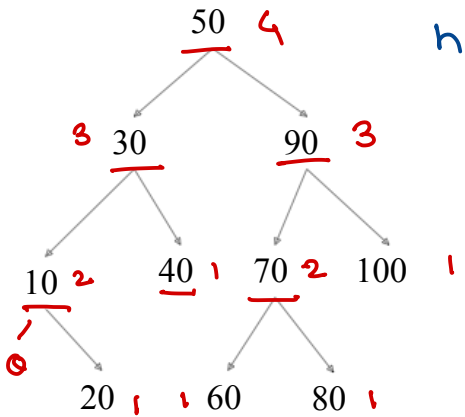
BST – Recursive Algorithm - del all




```
void delall(node* trav){
    if (trav == null)
        return;
    delall (trav->left);
    delall (trav->right);
    delete trav;
}
void delall() {
    delall(root);
    root = NULL;
}
~tree(){
    delall();
}
```

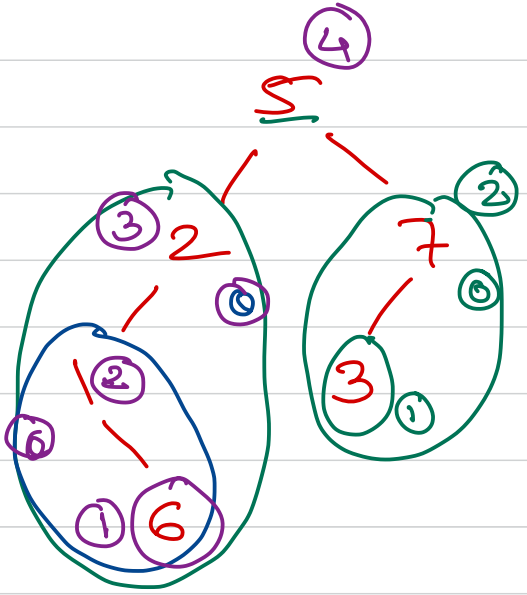


BST – Recursive Algorithm - height of node. & height of tree

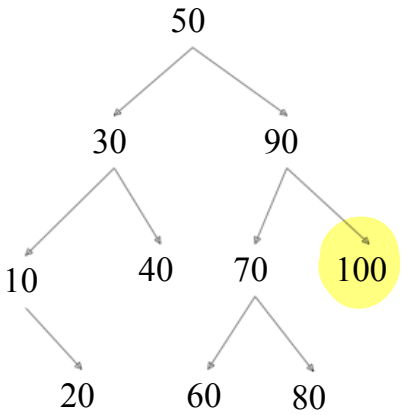


$h = \max(h(\text{left}), h(\text{right})) + 1$

```
int height(node *root){
    if(root==null)
        return 0;
    int hl= height(root->left);
    int hr= height(root->right);
    int max= hl>hr? hl:hr;
    return max+1;
}
```


4
5
3
2
0
2
6
1
6
7
2
0
3
1

BST – Non-Recursive Algorithm → Preorder V L R



start trav from root;

while trav is not null;
{ visit trav;
if trav has right,
push it on stack;
goto trav's left;
}
pop from stack & take
in trav

50 30 10 20 40
90 70 60 80 100

Thank you!

Nilesh Ghule <nilesh@sunbeaminfo.com>