CS 32 Week 8
Discussion 1I

Srinath

Outline

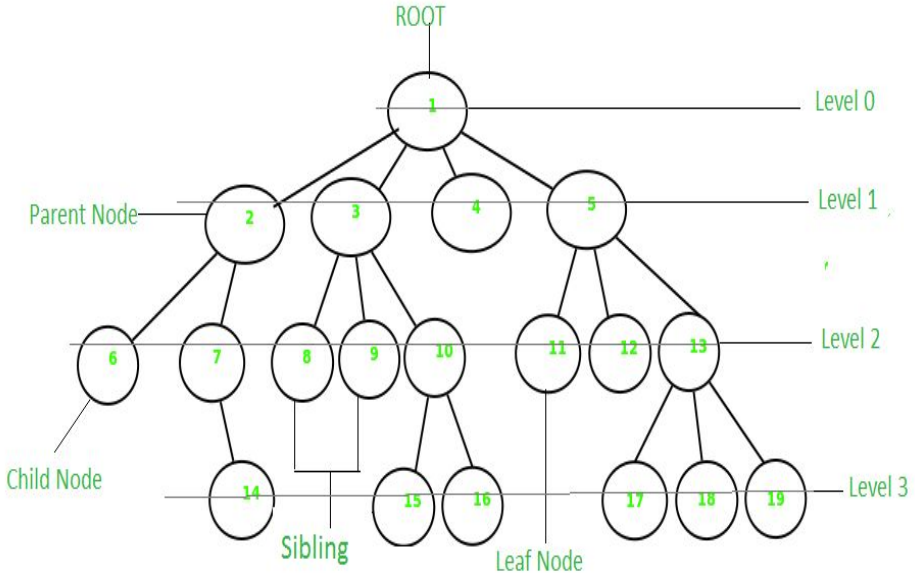
- Trees
- Binary Search Trees
- Worksheet 8

Trees

Trees :

Kind of data structure that holds hierarchy. Can be thought of as extension to linked lists

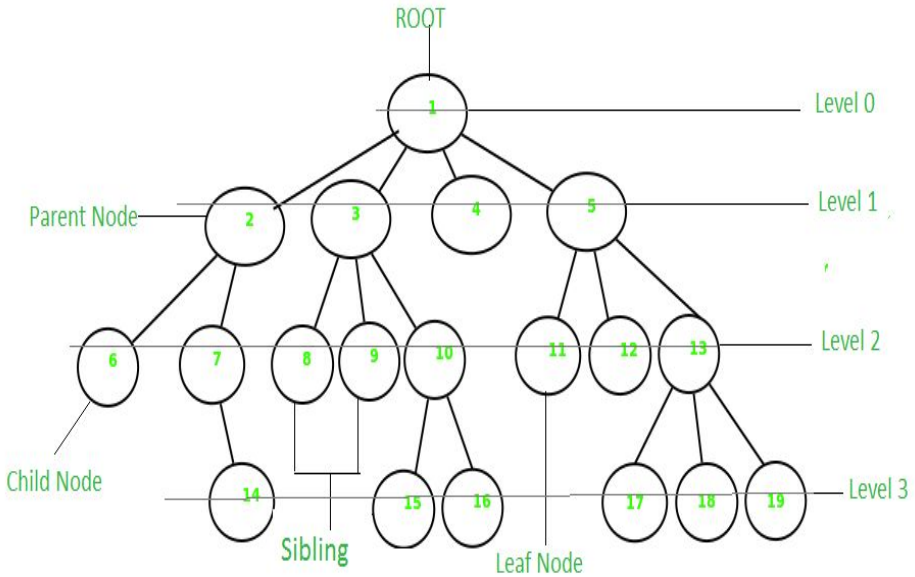
Terms : parent, child, root, leaf, sibling, height, subtree



Trees :

Kind of data structure that holds hierarchy. Can be thought of as extension to linked lists

Terms : parent, child, root, leaf, sibling, height, subtree

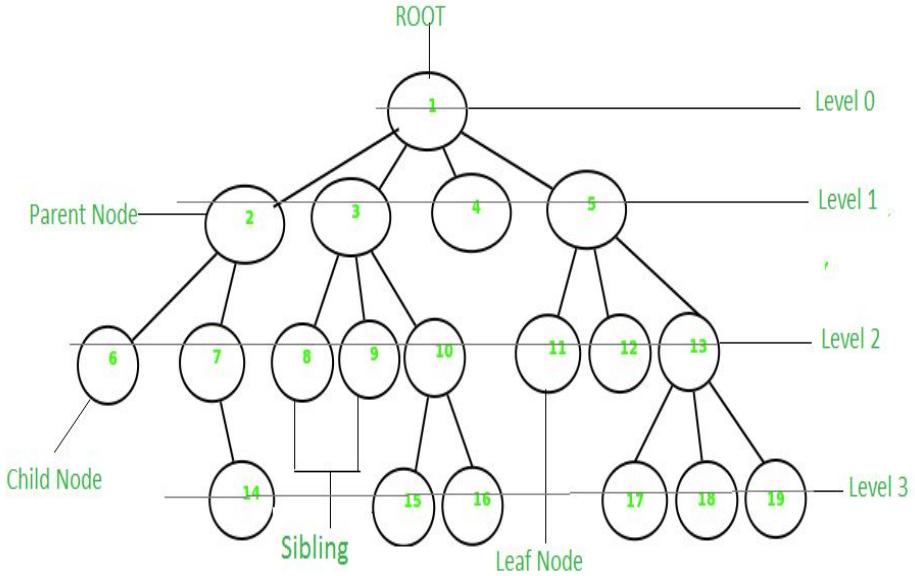


Code??

Trees :

Kind of data structure that holds hierarchy. Can be thought of as extension to linked lists

Terms : parent, child, root, leaf, sibling, height, subtree

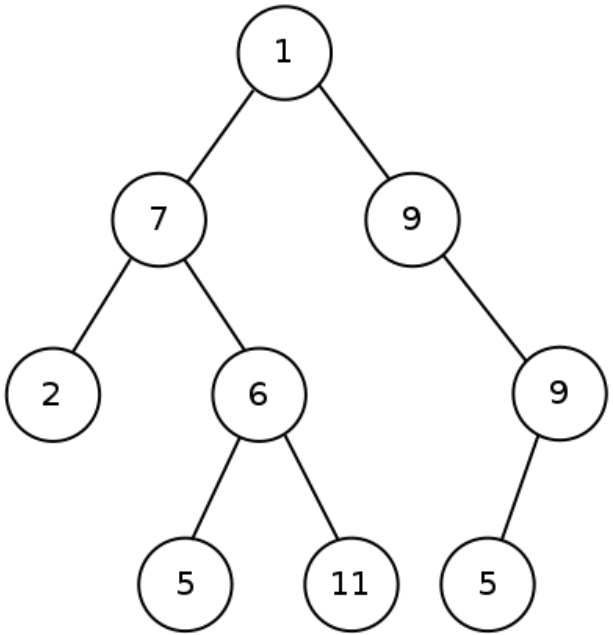


Code??

```
struct Node{
    int val;
    vector<Node*> children;
};

Node* root = new Node();
root→val = 100;

Node* root = NULL; // empty tree
```

Trees : Binary Tree

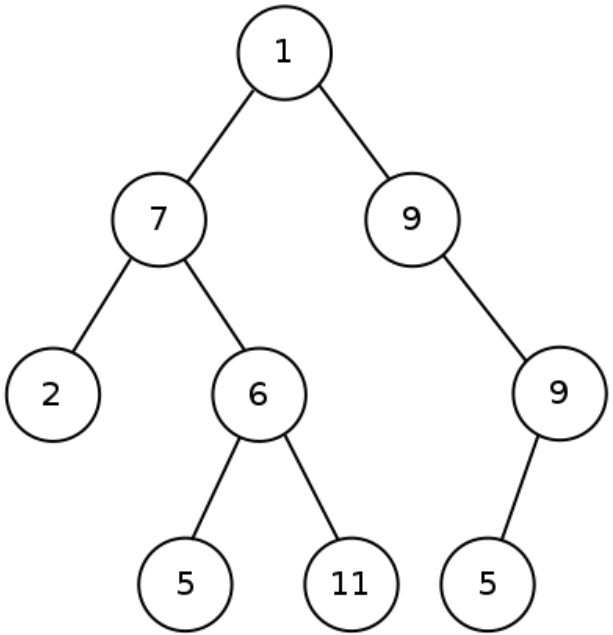
A simplified tree, each node has **at most 2** children.



Code??

Trees : Binary Tree

A simplified tree, each node has **at most 2** children.



Code??

```
struct Node{
    int val;
    Node* left;
    Node* right;
};

Node* root = new Node();
root→val = 100;

Node* root = NULL; // empty tree
```

Trees : Insertion

Insert a new value (n) at given Node* p

```
void insert(Node* p, int n){

}
```

Trees : Insertion

Insert a new value (n) at given Node* p

```
void insert(Node*  p, int n){
        // create a new node
        Node* myNode = new Node();
        myNode→val = n;
        if (p==NULL){ // empty tree
                p = myNode;
        }else{
            (p→children).push_back(myNode);
        }
        return;
}
```

```
Node * root = NULL;
insert(root, 190);
```

Trees : Insertion

Insert a new value (n) at given Node* p

```
void insert(Node*  p, int n){
        // create a new node
        Node* myNode = new Node();
        myNode→val = n;
        if (p==NULL){ // empty tree
                p = myNode;
        }else{
            (p→children).push_back(myNode);
        }
        return;
}
```

```
Node * root = NULL;
insert(root, 190);
```

Are we missing anything?

Trees : Insertion

Insert a new value (n) at given Node* p

```
void insert(Node* & p, int n){
        // create a new node
        Node* myNode = new Node();
        myNode→val = n;
        if (p==NULL){ // empty tree
                p = myNode;
        }else{
           (p→children).push_back(myNode);
        }
        return;
}
```

```
Node * root = NULL;
insert(root, 190);
```

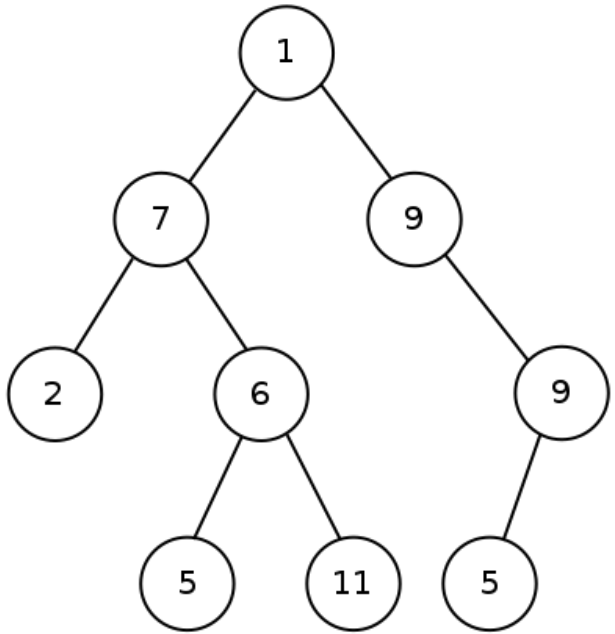
Are we missing anything?

- Yes, ‘root’ won’t be modified as we are passing by value

Trees : Traversal

To go through all nodes of a Tree.
Pre-Order : parent, then children
Post-Order : children, then parent
In-Order : left subtree, parent, right subtree (defined for binary trees)

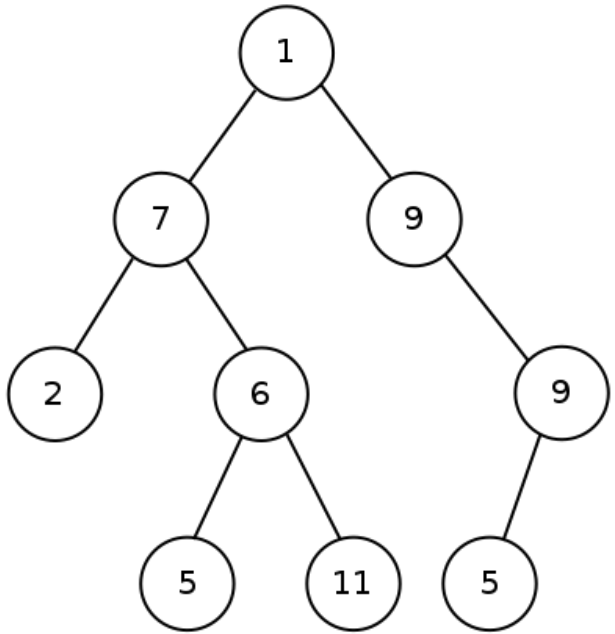
Pre-Order :
Post-Order :
In-Order :



Trees : Traversal

To go through all nodes of a Tree.
Pre-Order : parent, then children
Post-Order : children, then parent
In-Order : left subtree, parent, right subtree (defined for binary trees)

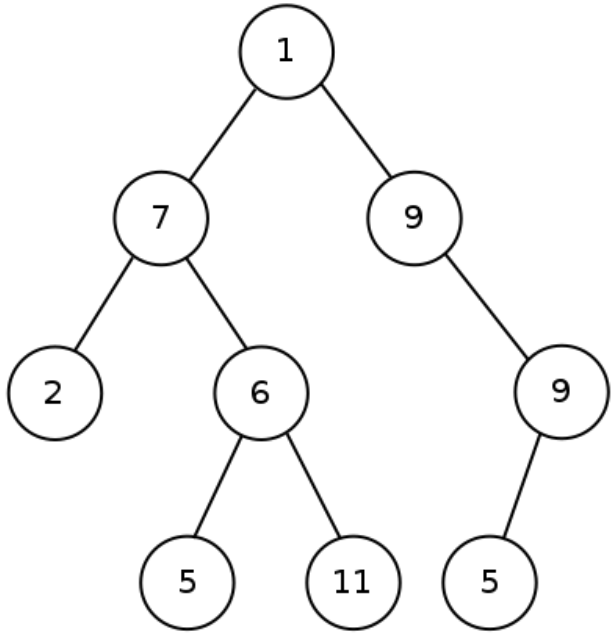
Pre-Order : 1, 7, 2, 6, 5, 11, 9, 9, 5
Post-Order : 2, 5, 11, 6, 7, 5, 9, 9, 1
In-Order : 2, 7, 5, 6, 11, 1, 9, 5, 9



Trees : Traversal

Code for pre-order traversal?

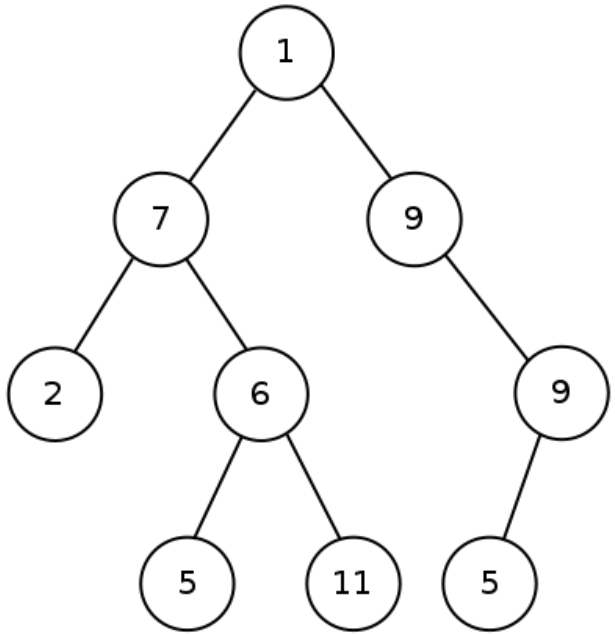
```
void pre-order-traversal(Node* root){

}
```



Trees : Traversal

Code for pre-order traversal?

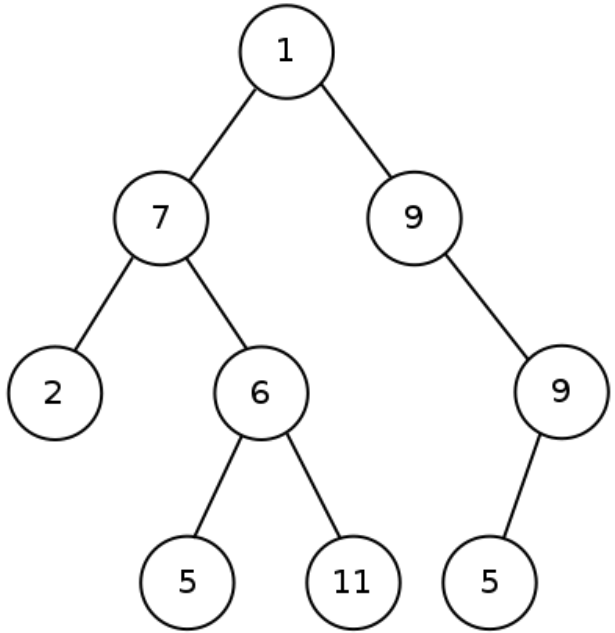
```
void pre-order-traversal(Node* root){
        // base case
        if(root == NULL) return;
        cout << root->val << " "; // curr node
        // then children
        for(int i=0; i<(root→children).size(); i++){
                pre-order-traversal((root→children)[i]);
        }
        return;
}
```



Trees : Traversal

Code for post-order traversal?

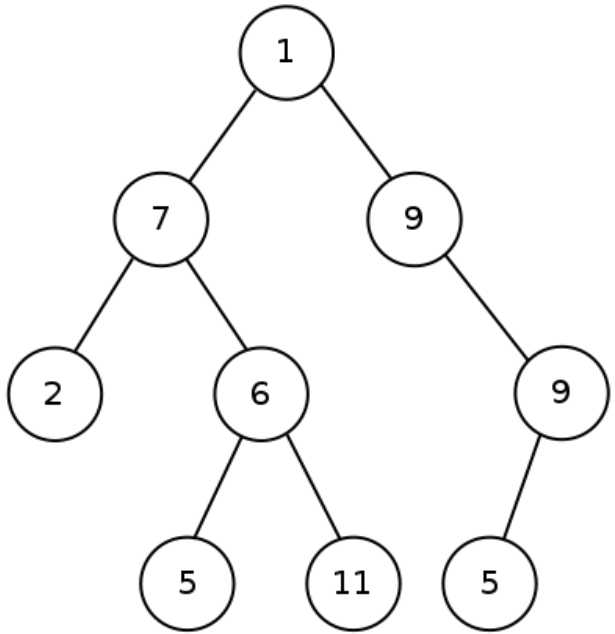
```
void post-order-traversal(Node* root){

}
```



Trees : Traversal

Code for post-order traversal?

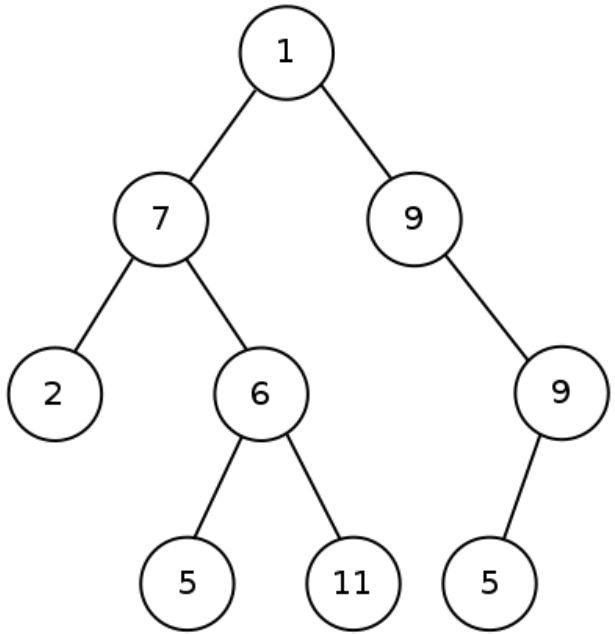
```
void post-order-traversal(Node* root){
        // base case
        if(root == NULL) return;
        // first children
        for(int i=0; i<(root→children).size(); i++){
                post-order-traversal((root→children)[i]);
        }
        // curr node
        cout << root->val << " ";
        return;
}
```



Trees : Traversal

Code for in-order traversal? (for a binary tree)

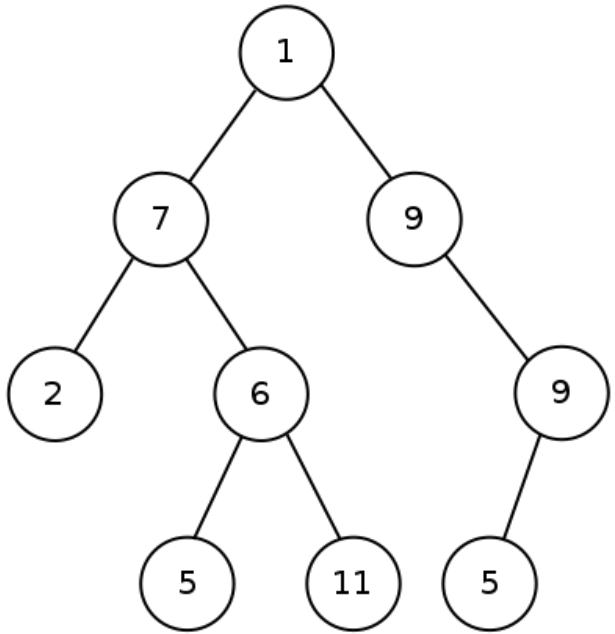
```
void in-order-traversal(Node* root){

}
```



Trees : Traversal

Code for in-order traversal? (for a binary tree)

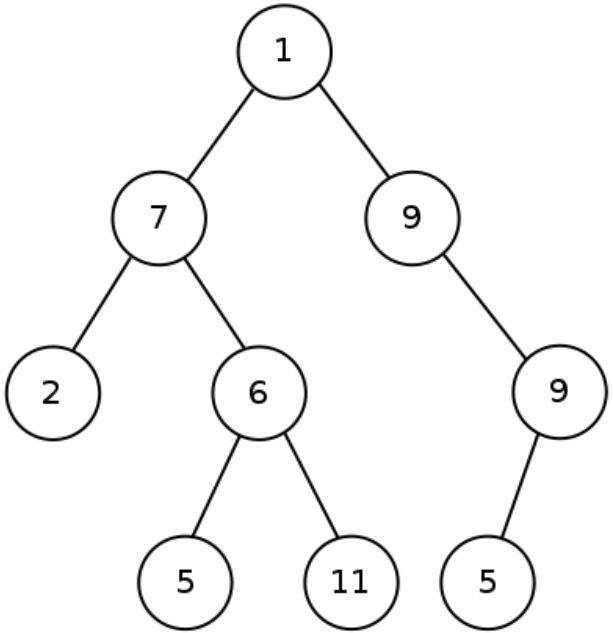
```
void in-order-traversal(Node* root){
        if(root == NULL) return;
        in-order-traversal(root→left);
        cout << root->val << " ";
        in-order-traversal(root→right);
}
```



Trees : Height

How to find height of a Tree?

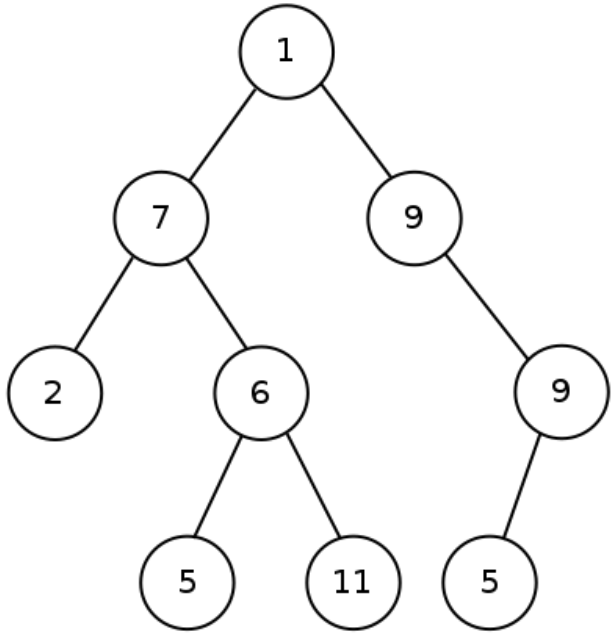
```
int height(Node* root){

}
```



Trees : Height

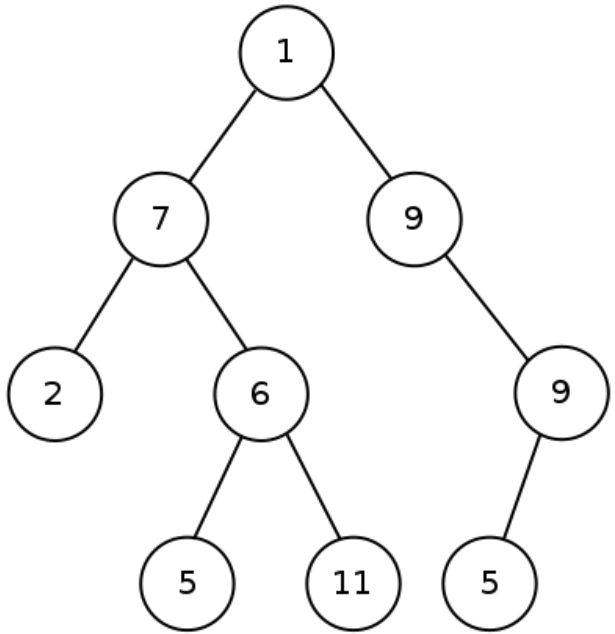
How to find height of a Tree?

```
int height(Node* root){
        if(root == NULL || (root->children).empty()) return 0;
        int maxChildHt = INT_MIN;
        for(int i=0; i<(root→children).size(); i++){
            maxChildHt = max(maxChildHt, height((root→children)[i]));
        }
        return 1+maxChildHt;
}
```



Trees : Deletion

How to delete a particular value (k) from tree.



Binary Search Trees (BST)

BST : Definition

Organised binary tree such that the following property holds

Given any node **p**,
Let **x** be any node in **left-subtree** of **p,**
Let **y** be any node in **right-subtree** of **p,**

then

x→val <= p→val <= y→val

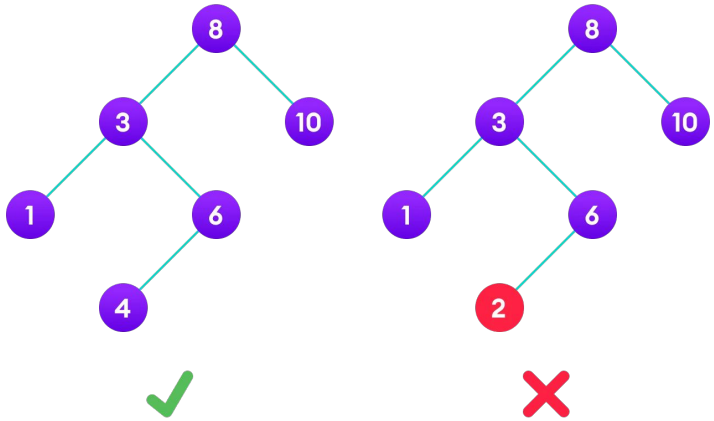
BST : Definition

Organised binary tree such that the following property holds

Given any node **p**,
Let **x** be any node in **left-subtree** of **p,**
Let **y** be any node in **right-subtree** of **p,**

then

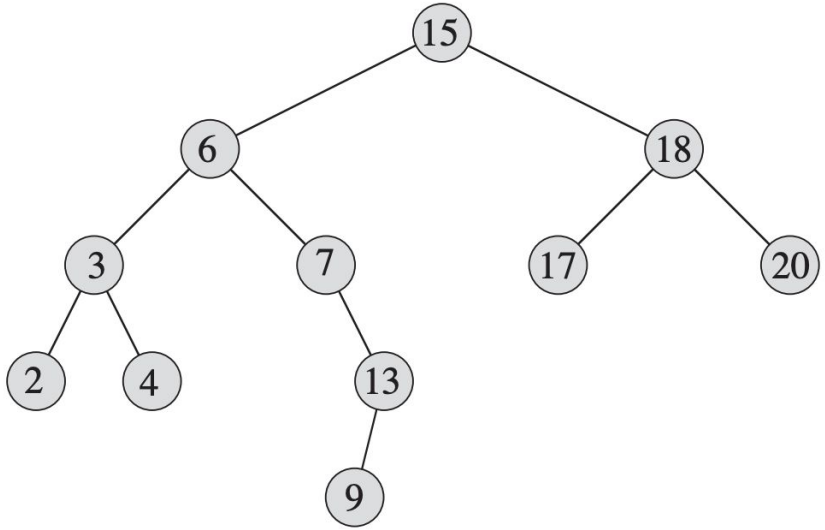
x→val <= p→val <= y→val



Src : https://cdn.programiz.com/sites/tutorial2program/files/bst-vs-not-bst.png

BST : Insertion

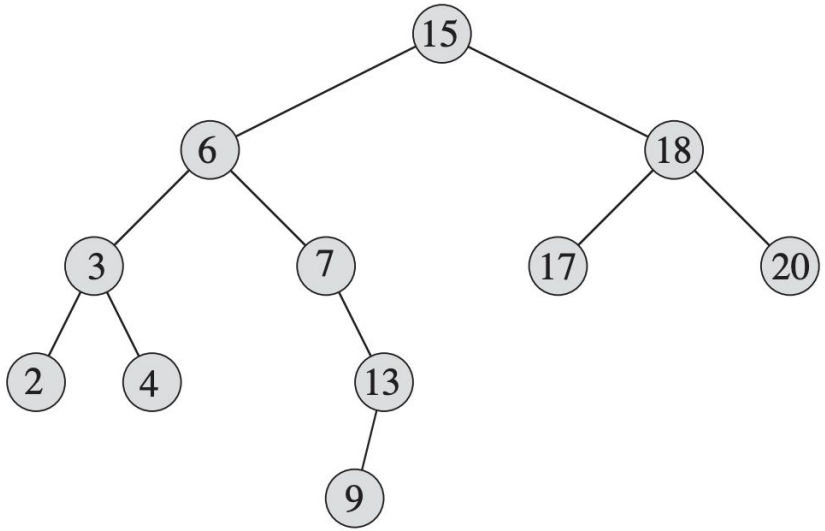
Insert a new value (n) in BST rooted at ‘root’
Insert should be in such a way that BST property is still valid after insertion.



BST : Insertion

Insert a new value (n) in BST rooted at ‘root’
Insert should be in such a way that BST property is still valid after insertion.

```
void insert(Node* & root, int n){
        if(root == NULL){
                root = new Node();
                root→val = n;
                root→left = NULL; root→right = NULL:
                return;
        }
        if(n < root→val){ // insert left
                insert(root→left, n);
        }else if (n > root→val){ // insert right
                        insert(root→right, n);
                }
                else{ // equal case (already exists)
                        return;
                }
}
```

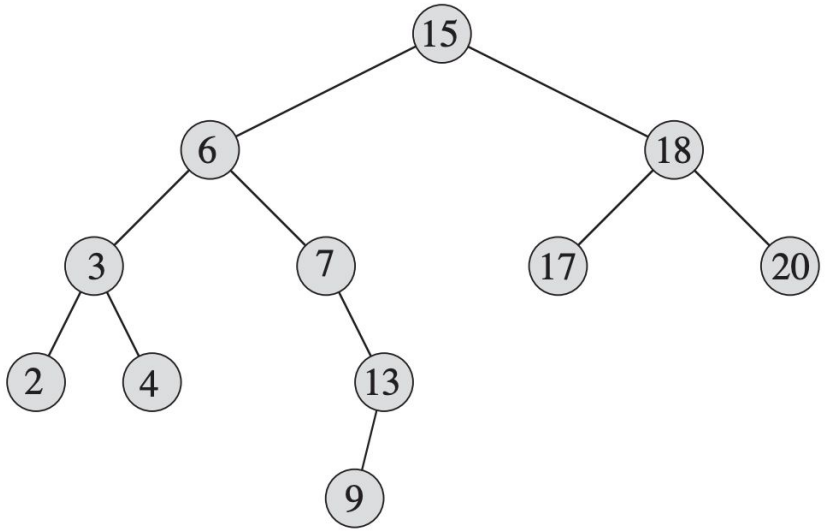


BST : Insertion

Insert a new value (n) in BST rooted at 'root'
Insert should be in such a way that BST property is still valid after insertion.

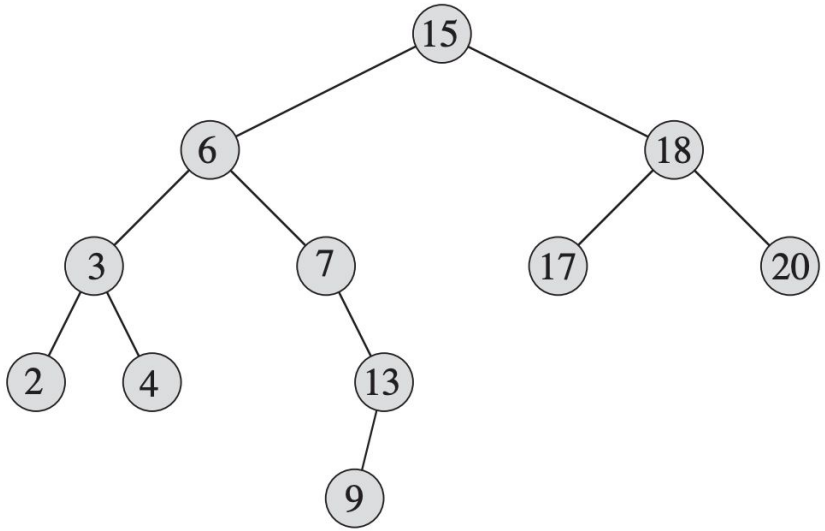
```
void insert(Node* & root, int n){
        if(root == NULL){
                root = new Node();
                root→val = n;
                root→left = NULL; root→right = NULL:
                return;
        }
        if(n < root→val){ // insert left
                insert(root→left, n);
        }else if (n > root→val){ // insert right
                        insert(root→right, n);
                }
                else{ // equal case (already exists)
                        return;
                }
}
```

Time Complexity : O(log N) - Average case



BST : Lookup | Search

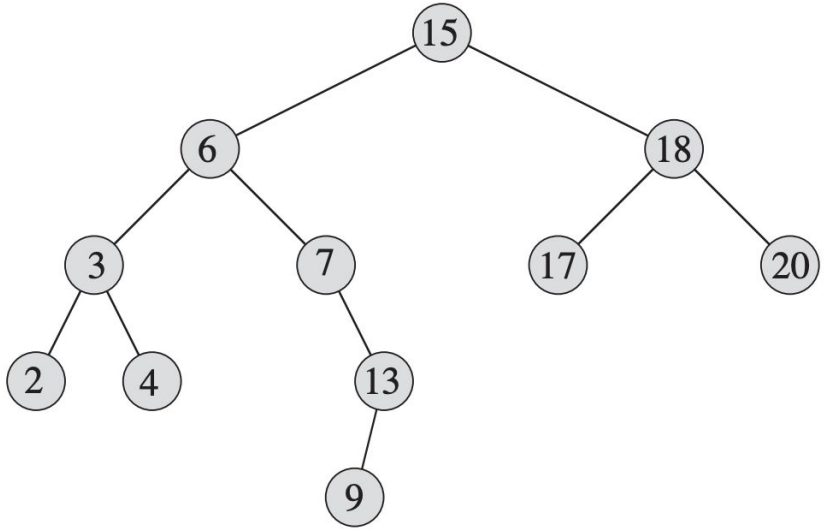
Does a given value (n) exist in BST rooted at ‘root’?



BST : Lookup | Search

Does a given value (n) exist in BST rooted at ‘root’?

```
bool search(Node* root, int n){
        if(root == NULL) return false;
        if(root→val == n) return true;
        if(n < root→val) return search(root→left, n);
        else return search(root→right, n);
}
```

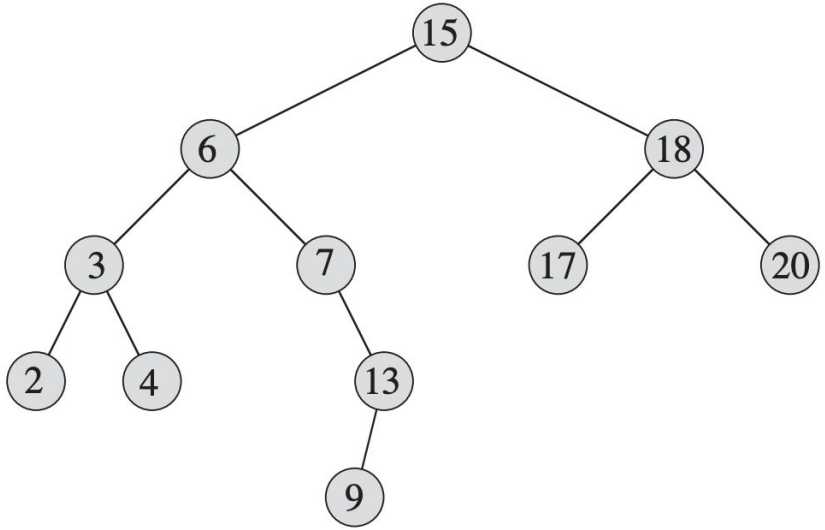


BST : Lookup | Search

Does a given value (n) exist in BST rooted at ‘root’?

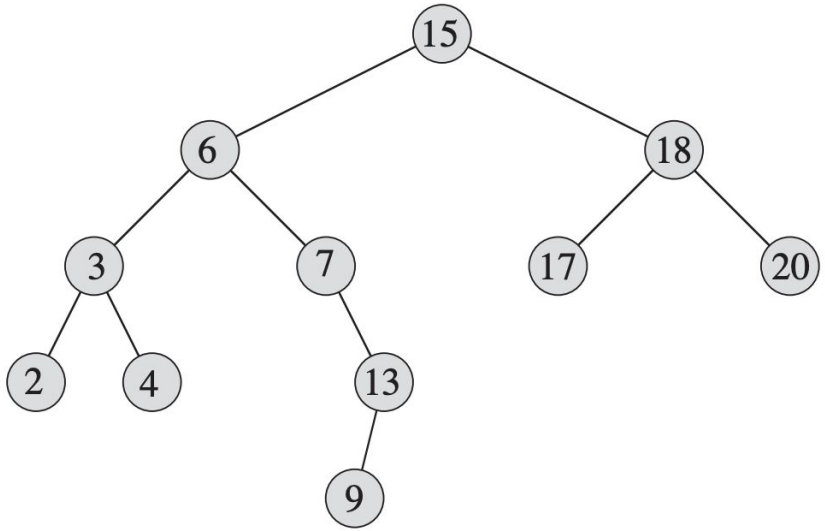
Time Complexity : O(log N) - Average case

```
bool search(Node* root, int n){
        if(root == NULL) return false;
        if(root→val == n) return true;
        if(n < root→val) return search(root→left, n);
        else return search(root→right, n);
}
```



BST : Traversal

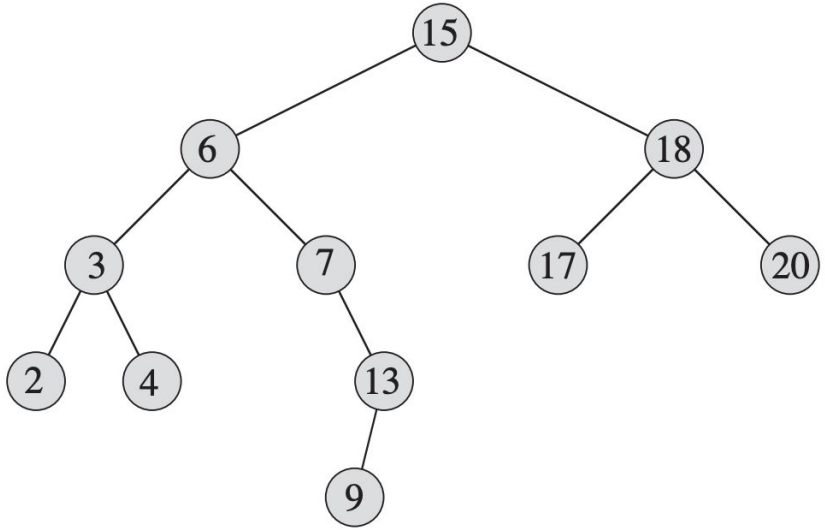
Pre-Order, Post-Order and In-Order
Similar to Binary Tree



BST : Traversal

Pre-Order, Post-Order and In-Order
Similar to Binary Tree

```
void in-order(Node* root){
        if(root == NULL) return;
        in-order(root→left);
        cout << root→val << " ";
        in-order(root→right);
}
```

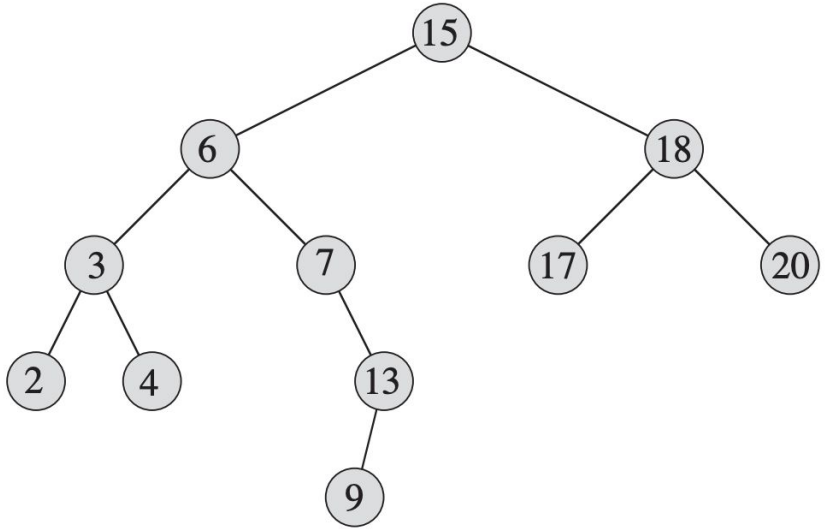


BST : Traversal

Pre-Order, Post-Order and In-Order
Similar to Binary Tree

Time Complexity : O(N)

```
void in-order(Node* root){
        if(root == NULL) return;
        in-order(root→left);
        cout << root→val << " ";
        in-order(root→right);
}
```



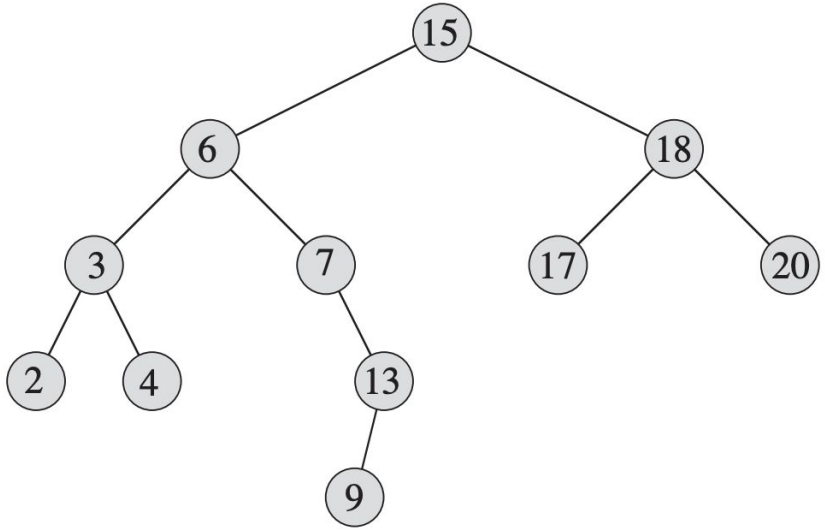
BST : Traversal

Pre-Order, Post-Order and In-Order
Similar to Binary Tree

What does in-order traversal print??

```
void in-order(Node* root){
        if(root == NULL) return;
        in-order(root→left);
        cout << root→val << " ";
        in-order(root→right);
}
```

Time Complexity : O(N)



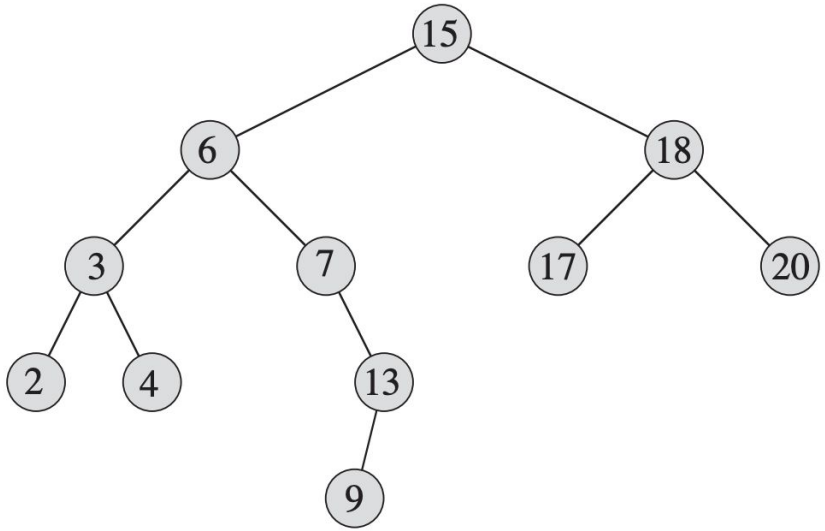
BST : Traversal

Pre-Order, Post-Order and In-Order
Similar to Binary Tree

What does in-order traversal print??
- Sorted order of elements

```
void in-order(Node* root){
        if(root == NULL) return;
        in-order(root→left);
        cout << root→val << " ";
        in-order(root→right);
}
```

Time Complexity : O(N)

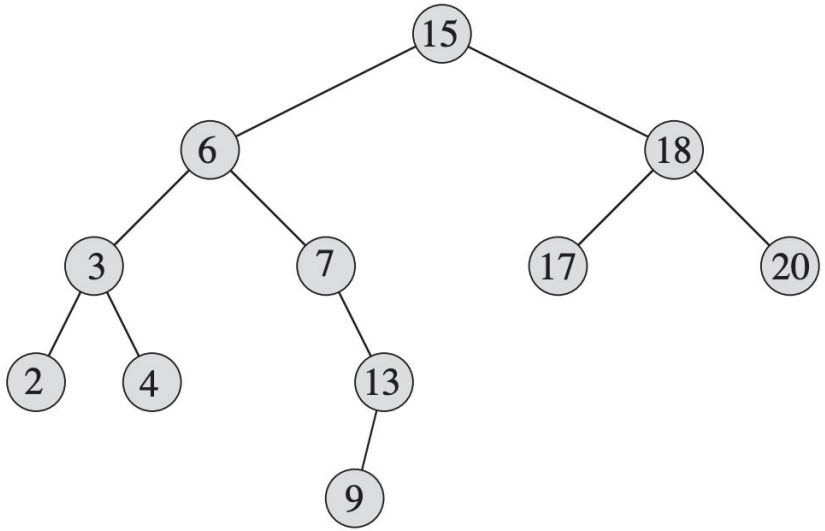


BST : Deletion

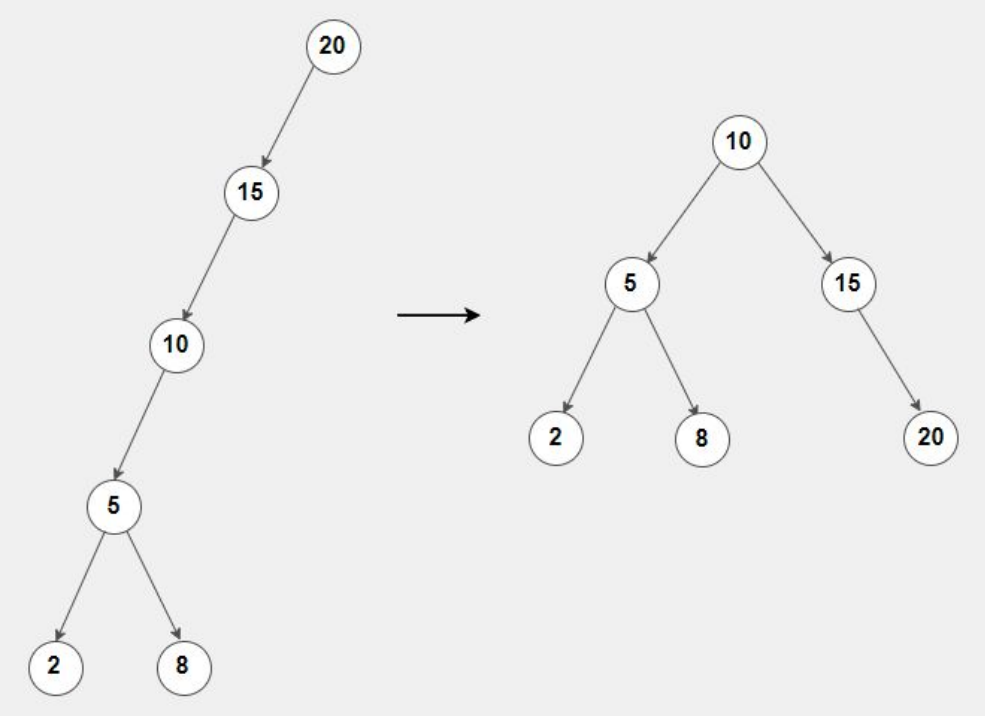
Delete a given value(k) from BST??

```
void delete(Node* root, int k){
}
```

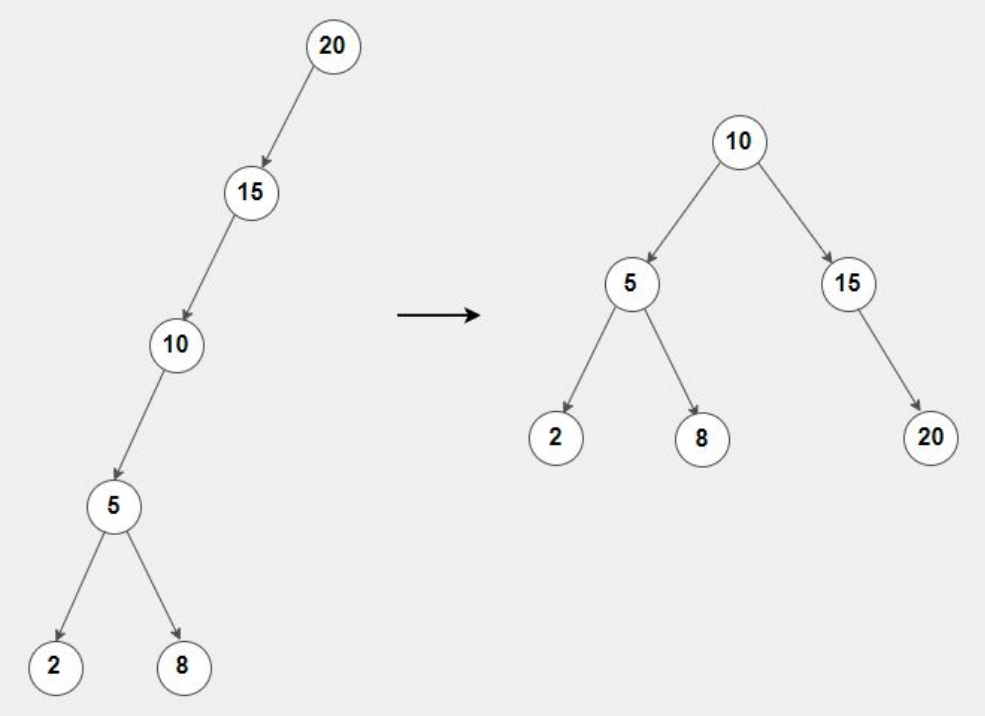
Time Complexity :



BST : Balanced vs Unbalanced



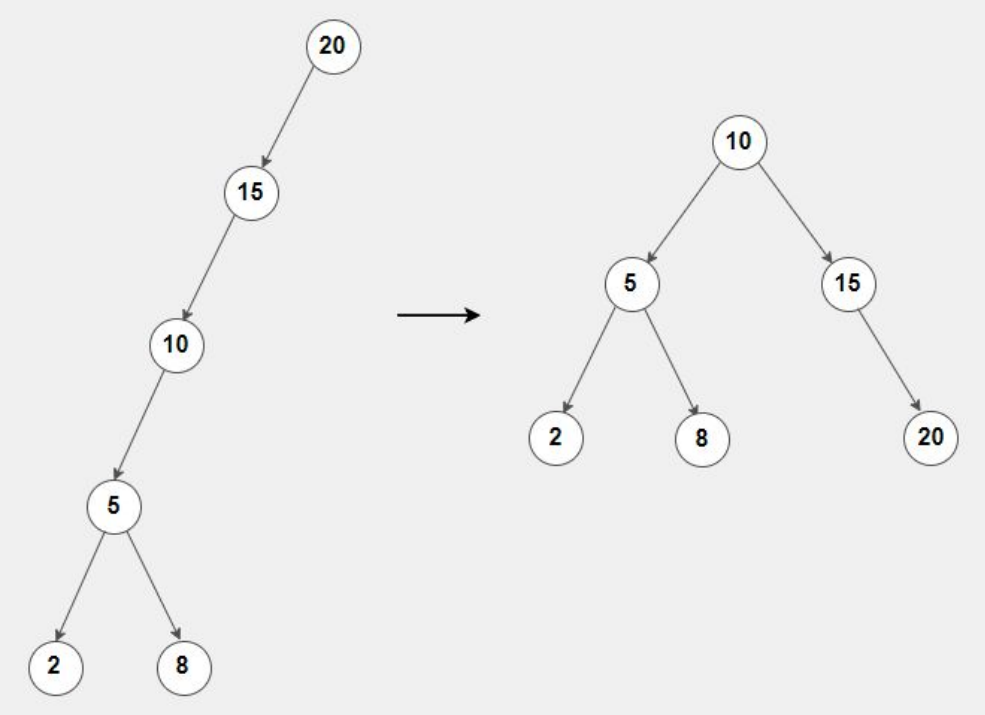
BST : Balanced vs Unbalanced



Balancing schemes

- AVL Tree
- Red-Black Tree
- 2-3 Tree
- ….. etc….

BST : Balanced vs Unbalanced



Balancing schemes

- AVL Tree
- Red-Black Tree
- 2-3 Tree
- ….. etc….

Basic operations on balanced ones has **O(log N)** complexity.

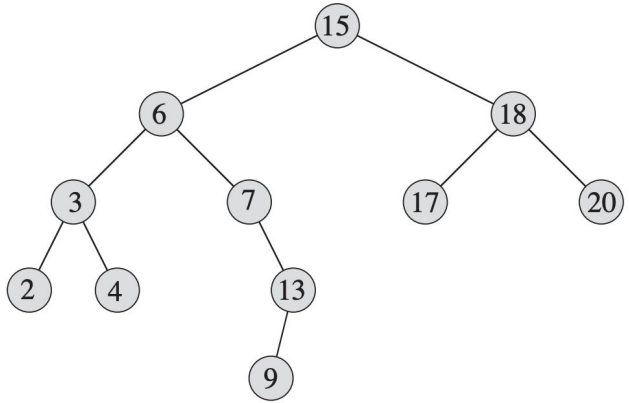
Unbalanced might go to worst case of **O(N)**

Keep in mind the **cost of balancing.**

BST : Problems

Find k'th smallest element in a BST??
BST Node is defined as

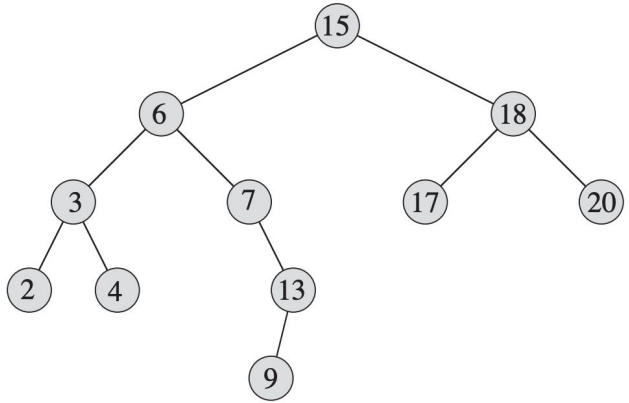
```
struct Node{
        int val;
        Node* left;
        Node* right;
        int m_size; //number of nodes in subtree.
}
```



BST : Problems

Find k'th smallest element in a BST??
BST Node is defined as

```
struct Node{
        int val;
        Node* left;
        Node* right;
        int m_size; //number of nodes in subtree.
}

int Ksmallest(Node* root, int k){
        int l_size = 0;
        if(root→left != NULL) l_size = root→left→m_size;
        if(l_size==k-1) return root→val;
        if(l_size < k-1) return Ksmallest(root→right, k-1- l_size);
        else return Ksmallest(root→left, k);
}
```

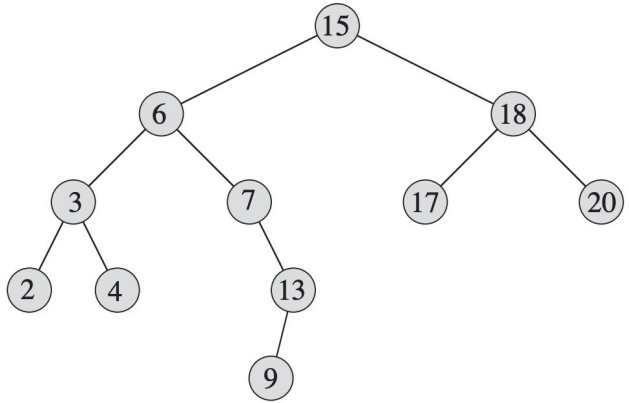


BST : Problems

Find k'th smallest element in a BST??
BST Node is defined as

```
struct Node{
        int val;
        Node* left;
        Node* right;
        int m_size; //number of nodes in subtree.
}

int Ksmallest(Node* root, int k){
        int l_size = 0;
        if(root→left != NULL) l_size = root→left→m_size;
        if(l_size==k-1) return root→val;
        if(l_size < k-1) return Ksmallest(root→right, k-1- l_size);
        else return Ksmallest(root→left, k);
}
```

Time Complexity : O(log N) - Average case



BST : Problems

Given N elements, what is time-complexity of converting them to BST structure ??

BST : Problems

Given N elements, what is time-complexity of converting them to BST structure ??

O(N log N)

How to find median of an input data stream ??

BST : Usage

STL **Set**, **Multiset** and **Map** use self-balancing BST.

```
#include <set> // includes both set and multiset

// declaration
set<type> s; // a set do not keep duplicates
multiset<type> ms; // a multiset can have duplicates

#include <map> // for map usage

// declaration
map<keyType, valueType> mp;
// stores paired data, (key, value)
// auto sorted by keys
// do not allow duplicate keys
```

refer www.cplusplus.com for detailed usage

References

BST
https://www.programiz.com/dsa/binary-search-tree

Content in some of the slides is taken from Yiyou Chen.