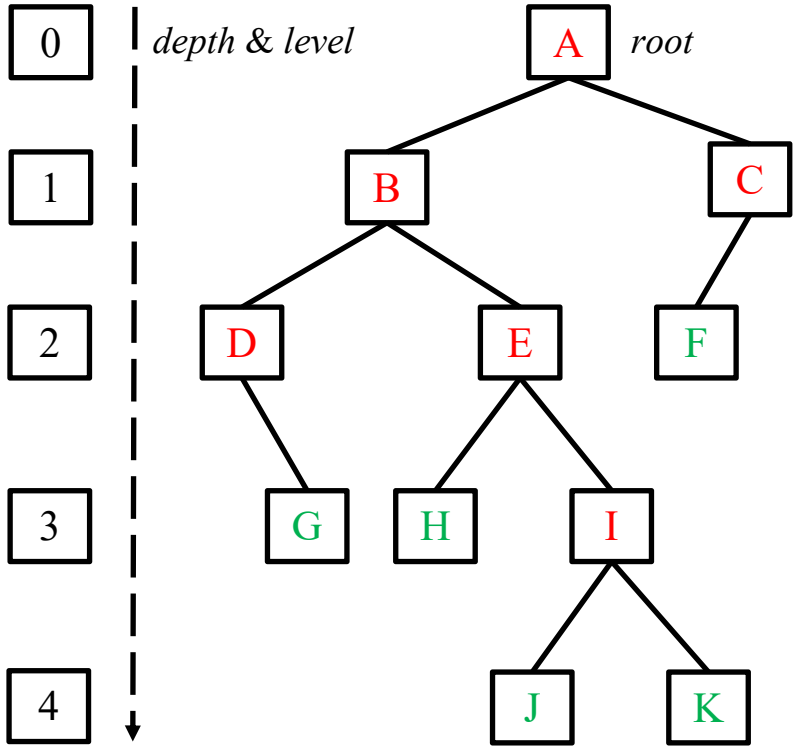
Binary Tree & Heap

LI Hao 李颢, Assoc. Prof. SPEIT & Dept. Automation of SEIEE

上海交通大學
SHANGHAI JIAO TONG UNIVERSITY

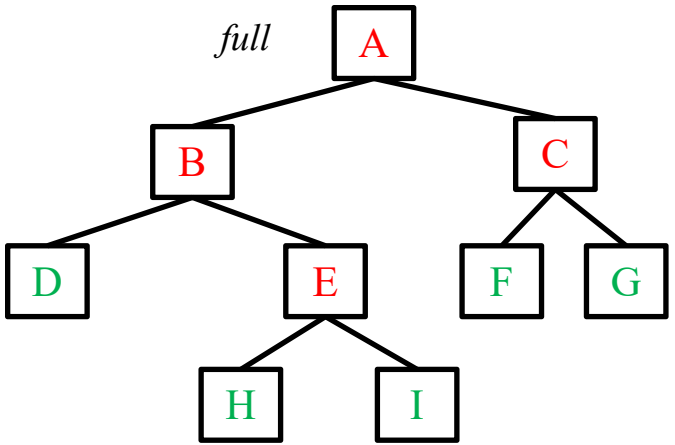
Binary Tree

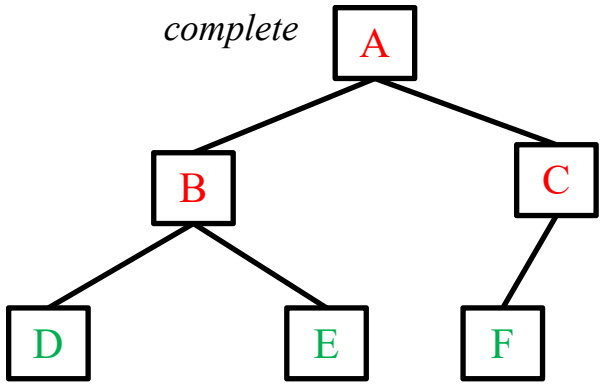
- **Binary tree**
  - *node* & *edge*
  - *root*
  - *subtree* (left & right)
  - *parent* & *children*
  - *ancestor* & *descendant*
  - *path* & *length*
    - B-E-I, A-C-F : two paths of length 2
    - A-B-E-I-J: a path of length 4
  - *depth* (cardinal) & *level* (ordinal)
  - *height* (largest depth+1)
  - *leaf node* & *internal node*



Binary Tree

- **Binary tree**
  - *full*
    - each node is either 1) an internal node with exactly two non-empty children, or 2) a leaf
  - *complete*
    - starting at the root and filled by levels from left to right

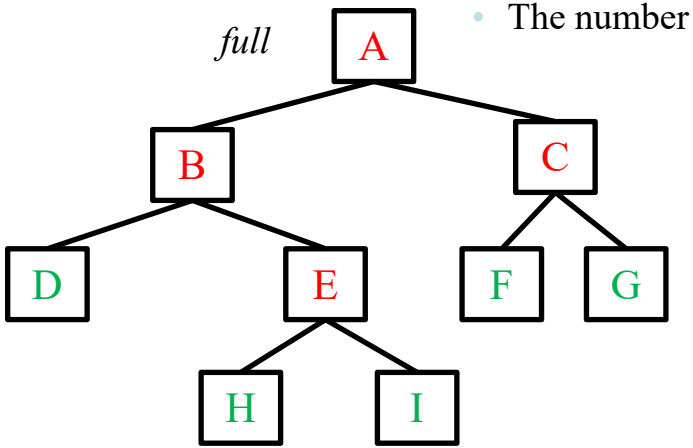


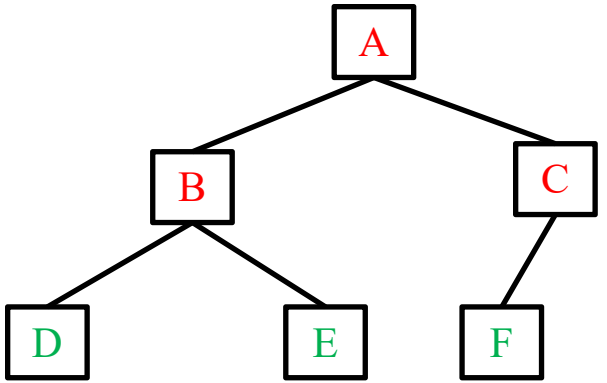


Binary Tree

- **Binary tree**
  - *full*
    - each node is either 1) an internal node with exactly two non-empty children, or 2) a leaf
  - *Full binary tree theorem*
    - The number of leaves in a non-empty full BT is one more than the number of internal nodes
  - *Corollary*
    - The number of empty subtrees in a non-empty BT is one more than the number of BT nodes

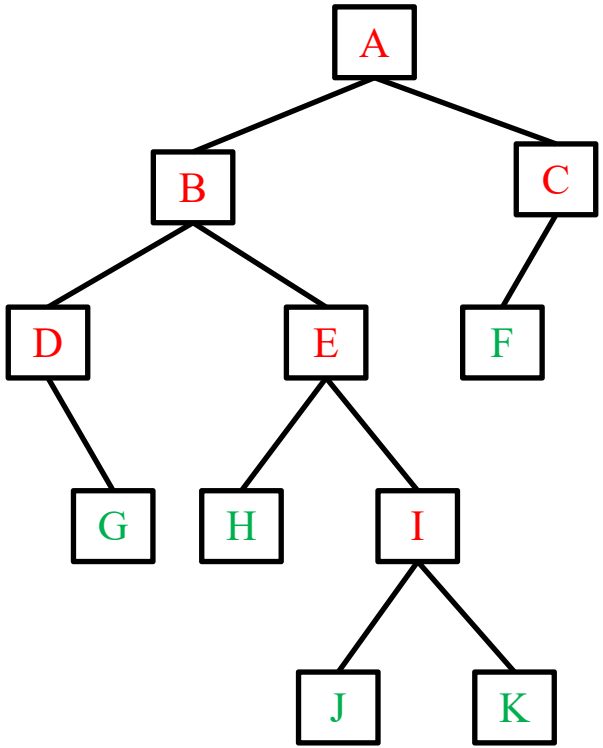
full





Binary Tree

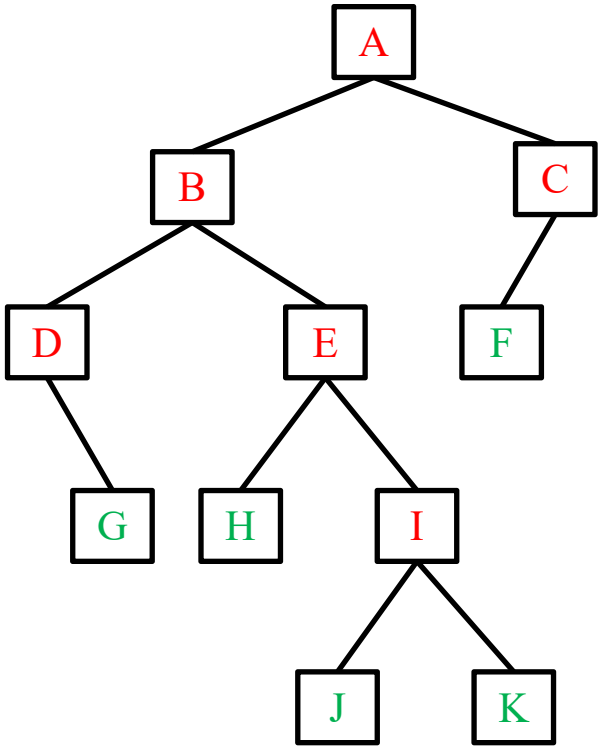
- **Binary tree traversal**
  - *enumeration*
    - a traversal listing every node exactly once
  - *preorder traversal*
    - visit any node *before* visiting its children
    - ABDGEHIJKCF
  - *postorder traversal*
    - visit any node *after* visiting its descendants
    - GDHJKIEBFCA
  - *inorder traversal*
    - visit the left subtree, then the node, finally the right subtree
    - DGBHEJIKAFC



Binary Tree

structure abstraction

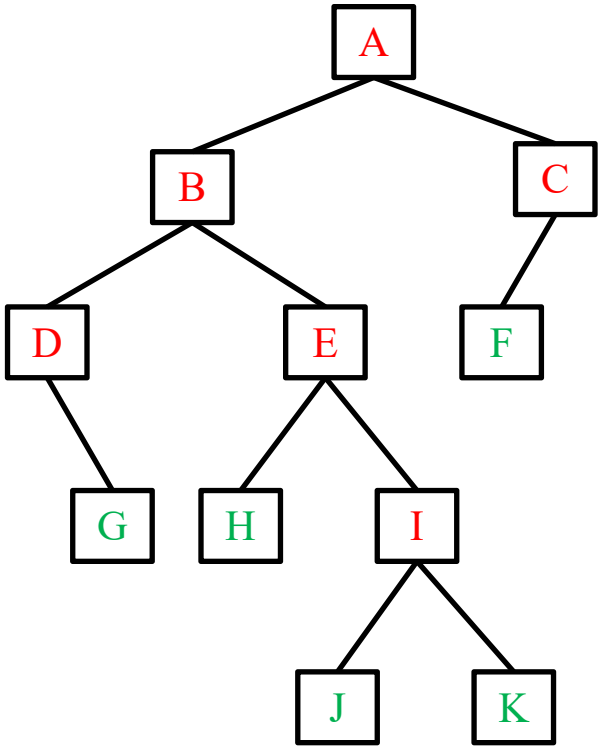
- **Binary tree node**




```
#ifndef __BNODE_H__
#define __BNODE_H__
template <typename T> class BNode{ // Binary tree node (ADT)
public: virtual ~BNode(){} // base destructor
        virtual T& getE()=0; // return node's value
        virtual void setE(const T&)=0; // set node's value
        virtual BNode* getL() const=0; // return node's left child
        virtual void setL(BNode*)=0; // set node's left child
        virtual BNode* getR() const=0; // return node's right child
        virtual void setR(BNode*)=0; // set node's right child
        virtual bool isLeaf()=0; // check if being a leaf
};
#endif
```

Binary Tree

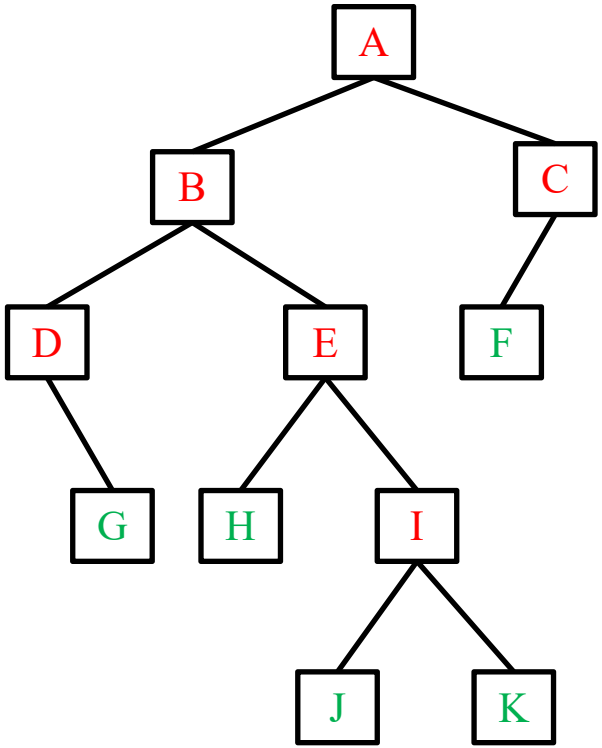
- **Binary tree node**




```
#include <iostream>
#include "BNode.h"
template <typename T> class BTNode: public BNode<T>{
private:T e; // node's value
        BTNode* cL; BTNode* cR; // node's left child & right child
public: BTNode(){cL=cR=NULL;} // constructor without initial values
        BTNode(const T& ei,BTNode* L=NULL,BTNode* R=NULL){
                e=ei;cL=L;cR=R;} // constructor with initial values
        ~BTNode(){}
        T& getE(){return e;}
        void setE(const T& ei){e=ei;}
        inline BTNode* getL() const{return cL;}
        void setL(BNode<T>* b){cL=(BTNode*)b;}
        inline BTNode* getR() const{return cR;}
        void setR(BNode<T>* b){cR=(BTNode*)b;}
        bool isLeaf(){return (cL==NULL)&&(cR==NULL);}
};
```

Binary Tree

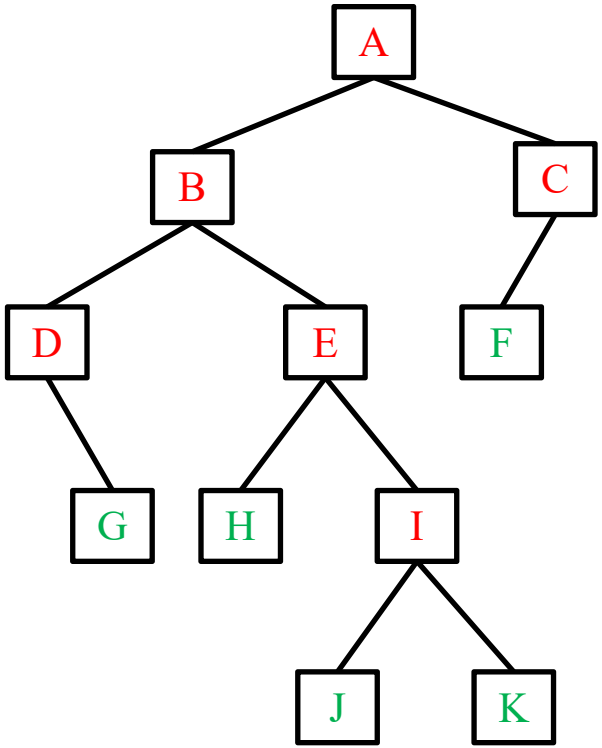
- **Binary tree traversal**




```
template <typename T> void preorder(BNode<T>* root){
        if (root==NULL) return;
        std::cout<<' '<<root->getE();
        preorder(root->getL());
        preorder(root->getR());
}
template <typename T> void postorder(BNode<T>* root){
        if (root==NULL) return;
        postorder(root->getL());
        postorder(root->getR());
        std::cout<<' '<<root->getE();
}
template <typename T> void inorder(BNode<T>* root){
        if (root==NULL) return;
        inorder(root->getL());
        std::cout<<' '<<root->getE();
        inorder(root->getR());
}
```

Binary Tree

- **Binary tree traversal**



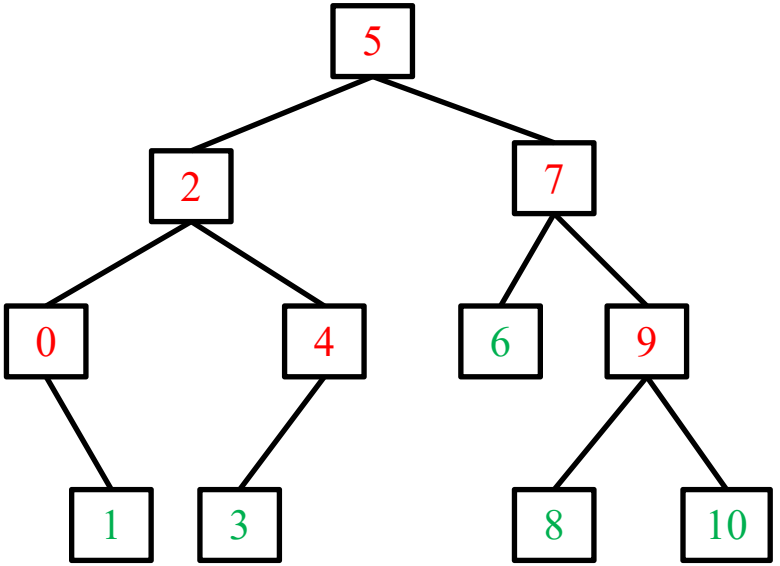

```
g++ demoBT.cpp -o _a; ./_a; rm _a
Preorder:  A B D G E H I J K C F
Postorder:  G D H J K I E B F C A
Inorder:  D G B H E J I K A F C
```

```
#include <iostream>
#include "BTNode.h"
using namespace std;

int main(){
        BTNode<char> J('J'),K('K'),I('I',&J,&K),H('H'),G('G'),
                F('F'),D('D',NULL,&G),E('E',&H,&I),B('B',&D,&E),
                C('C',&F,NULL),A('A',&B,&C);
        cout<<"Preorder: ";preorder(&A);cout<<endl;
        cout<<"Postorder: ";postorder(&A);cout<<endl;
        cout<<"Inorder: ";inorder(&A);cout<<endl;
        return 0;
}
```

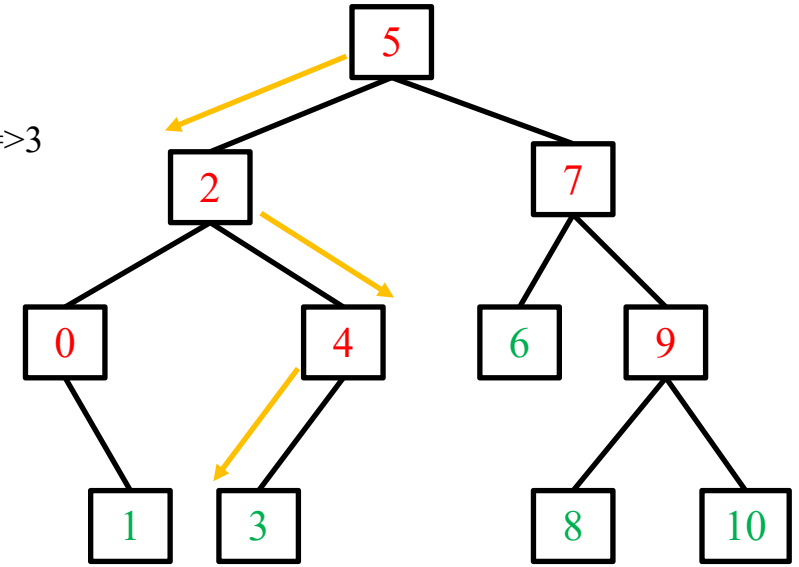
Binary Search Tree

- **Binary search tree property**
  - left subtree with key values < self key value
  - right subtree with key values ≥ self key value



Binary Search Tree

- **Binary search tree property**
  - left subtree with key values < self key value
  - right subtree with key values ≥ self key value
- **Binary search**
  - search down only one subtree
    - e.g. find 3 : 5(3<5)=>2(3>2)=>4(3<4)=>3

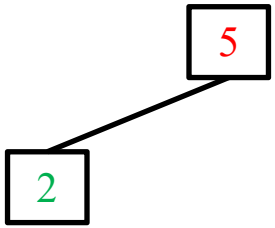


Binary Search Tree

- **Binary search tree property**
  - left subtree with key values < self key value
  - right subtree with key values ≥ self key value
- **Binary search**
  - search down only one subtree
    - e.g. find 3 : 5(3<5)=>2(3>2)=>4(3<4)=>3
- **Insert new nodes**
  - binary search & insert
    - 5>2>4>3>7>6>0>1>9>10>8

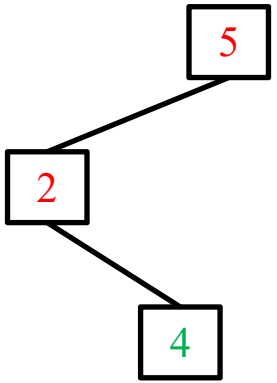
Binary Search Tree

- **Binary search tree property**
  - left subtree with key values < self key value
  - right subtree with key values ≥ self key value
- **Binary search**
  - search down only one subtree
    - e.g. find 3 : 5(3<5)=>2(3>2)=>4(3<4)=>3
- **Insert new nodes**
  - binary search & insert
    - 5>2>4>3>7>6>0>1>9>10>8



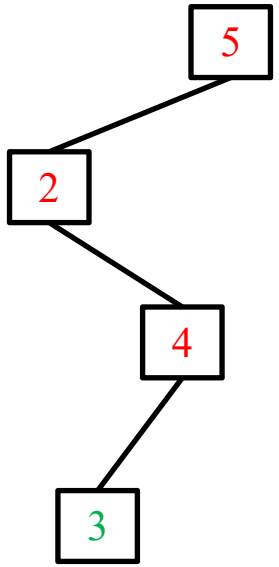
Binary Search Tree

- **Binary search tree property**
  - left subtree with key values < self key value
  - right subtree with key values ≥ self key value
- **Binary search**
  - search down only one subtree
    - e.g. find 3 : 5(3<5)=>2(3>2)=>4(3<4)=>3
- **Insert new nodes**
  - binary search & insert
    - 5>2>4>3>7>6>0>1>9>10>8



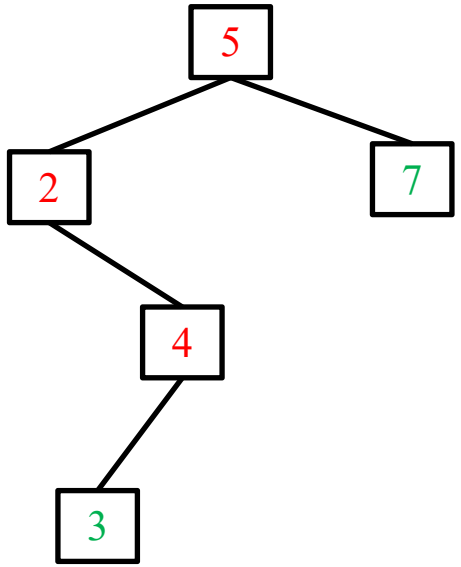
Binary Search Tree

- **Binary search tree property**
  - left subtree with key values < self key value
  - right subtree with key values ≥ self key value
- **Binary search**
  - search down only one subtree
    - e.g. find 3 : 5(3<5)=>2(3>2)=>4(3<4)=>3
- **Insert new nodes**
  - binary search & insert
    - 5>2>4>3>7>6>0>1>9>10>8



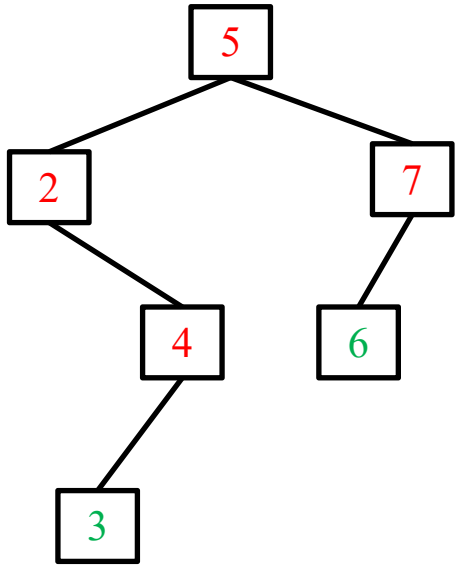
Binary Search Tree

- **Binary search tree property**
  - left subtree with key values < self key value
  - right subtree with key values ≥ self key value
- **Binary search**
  - search down only one subtree
    - e.g. find 3 : 5(3<5)=>2(3>2)=>4(3<4)=>3
- **Insert new nodes**
  - binary search & insert
    - 5>2>4>3>7>6>0>1>9>10>8



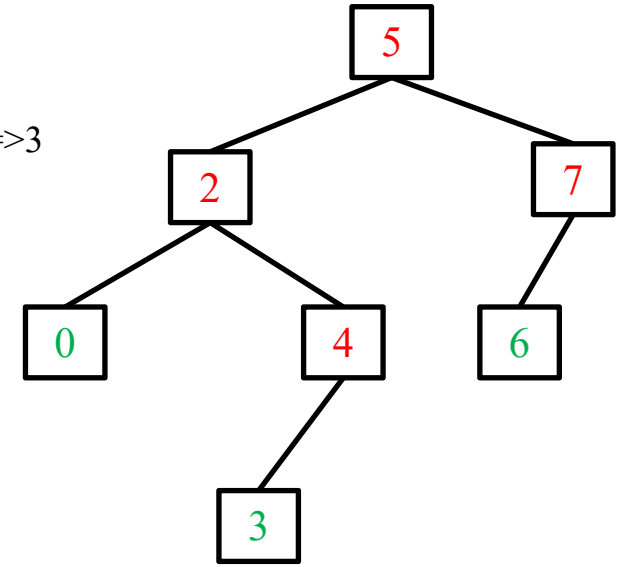
Binary Search Tree

- **Binary search tree property**
  - left subtree with key values < self key value
  - right subtree with key values ≥ self key value
- **Binary search**
  - search down only one subtree
    - e.g. find 3 : 5(3<5)=>2(3>2)=>4(3<4)=>3
- **Insert new nodes**
  - binary search & insert
    - 5>2>4>3>7>6>0>1>9>10>8



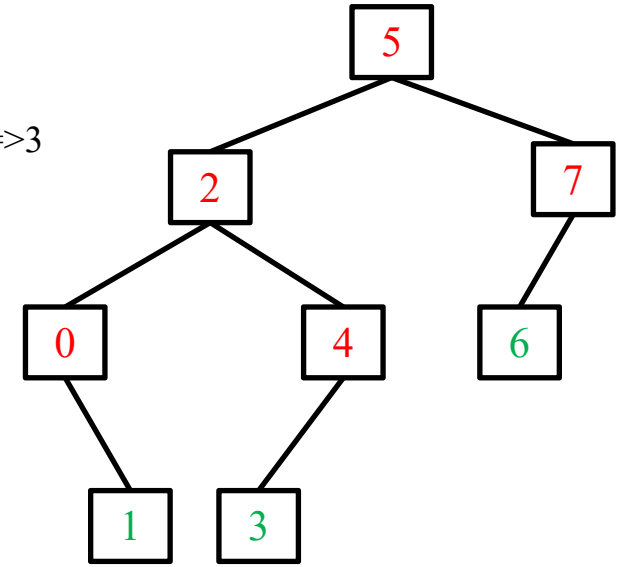
Binary Search Tree

- **Binary search tree property**
  - left subtree with key values < self key value
  - right subtree with key values ≥ self key value
- **Binary search**
  - search down only one subtree
    - e.g. find 3 : 5(3<5)=>2(3>2)=>4(3<4)=>3
- **Insert new nodes**
  - binary search & insert
    - 5>2>4>3>7>6>0>1>9>10>8



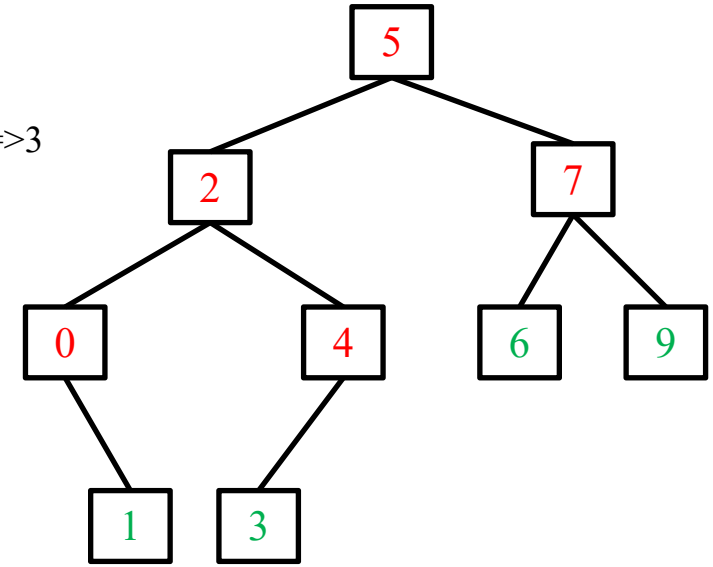
Binary Search Tree

- **Binary search tree property**
  - left subtree with key values < self key value
  - right subtree with key values ≥ self key value
- **Binary search**
  - search down only one subtree
    - e.g. find 3 : 5(3<5)=>2(3>2)=>4(3<4)=>3
- **Insert new nodes**
  - binary search & insert
    - 5>2>4>3>7>6>0>1>9>10>8



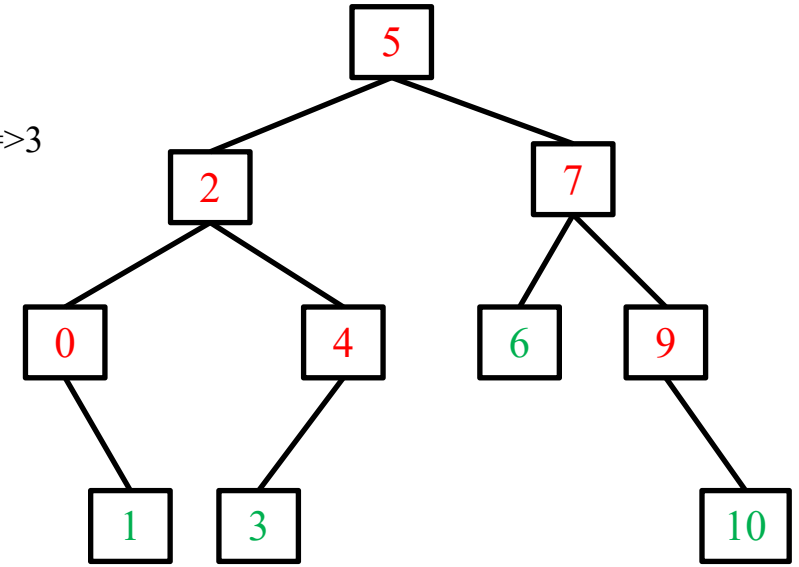
Binary Search Tree

- **Binary search tree property**
  - left subtree with key values < self key value
  - right subtree with key values ≥ self key value
- **Binary search**
  - search down only one subtree
    - e.g. find 3 : 5(3<5)=>2(3>2)=>4(3<4)=>3
- **Insert new nodes**
  - binary search & insert
    - 5>2>4>3>7>6>0>1>9>10>8



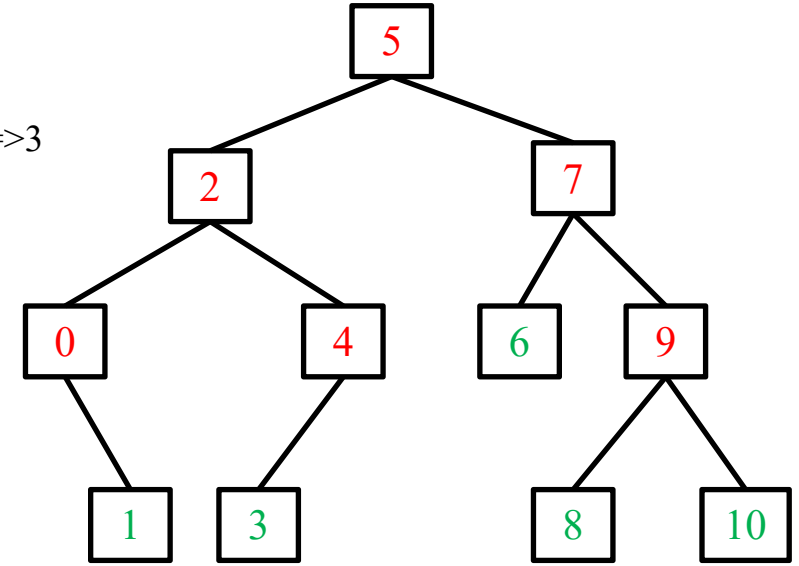
Binary Search Tree

- **Binary search tree property**
  - left subtree with key values < self key value
  - right subtree with key values ≥ self key value
- **Binary search**
  - search down only one subtree
    - e.g. find 3 : 5(3<5)=>2(3>2)=>4(3<4)=>3
- **Insert new nodes**
  - binary search & insert
    - 5>2>4>3>7>6>0>1>9>10>8



Binary Search Tree

- **Binary search tree property**
  - left subtree with key values < self key value
  - right subtree with key values ≥ self key value
- **Binary search**
  - search down only one subtree
    - e.g. find 3 : 5(3<5)=>2(3>2)=>4(3<4)=>3
- **Insert new nodes**
  - binary search & insert
    - 5>2>4>3>7>6>0>1>9>10>8



THANK YOU

上海交通大學
SHANGHAI JIAO TONG UNIVERSITY