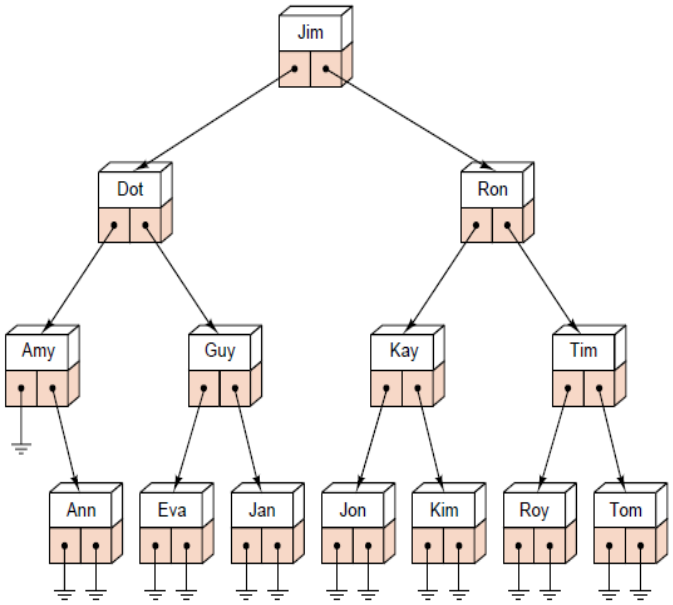
CS214-Data Structure

Binary Search Tree

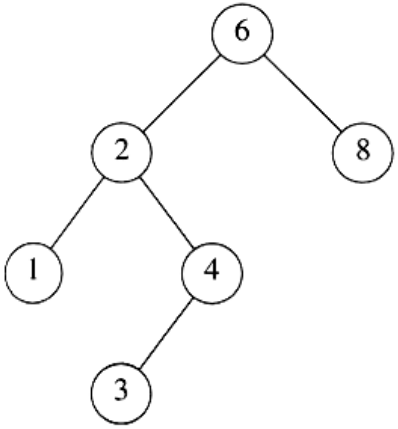
Binary Search Trees

A ***binary search tree is a binary tree that is either empty or in which every node*** has a key (within its data entry) and satisfies the following conditions:

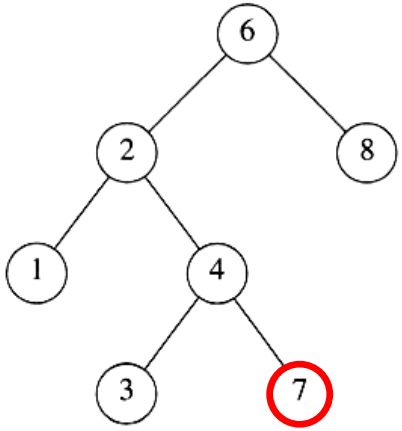
1. The key of the root (if it exists) is greater than the key in any node in the left subtree of the root.
2. The key of the root (if it exists) is less than the key in any node in the right subtree o the root.
3. The left and right subtrees of the root are again binary search trees.



Binary Search Trees



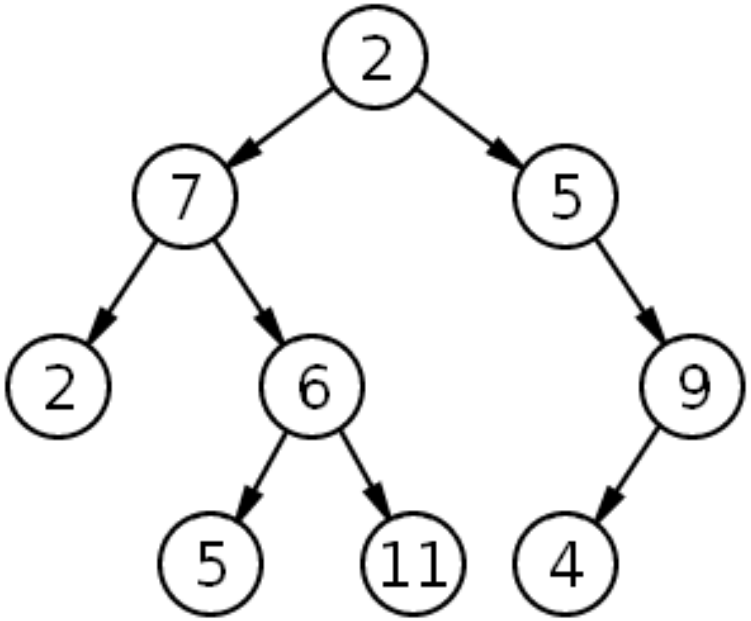
A binary search tree



Not a binary search tree

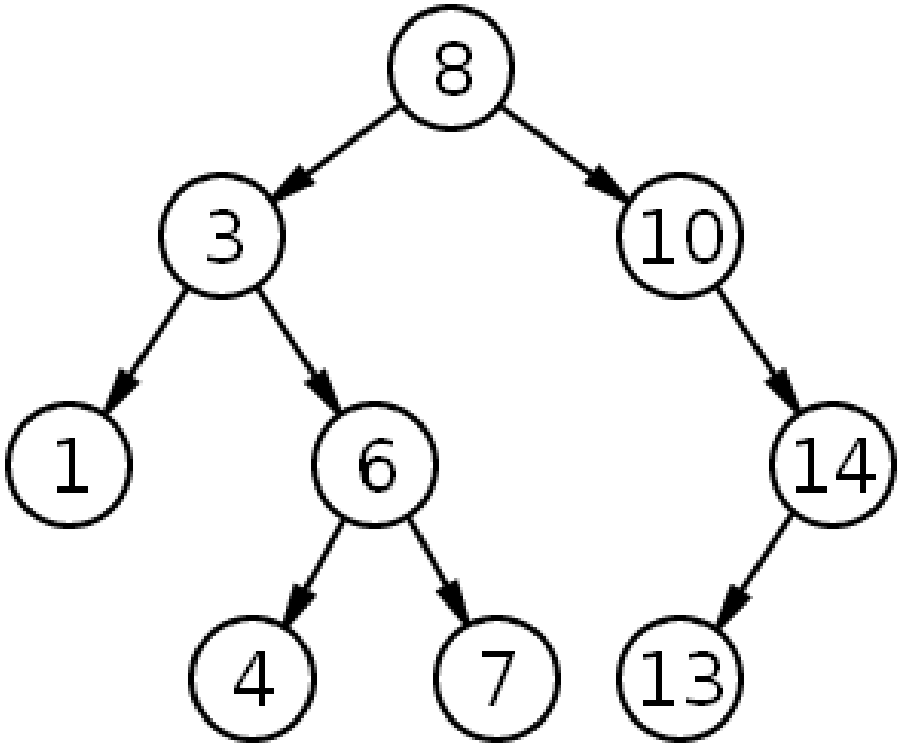
Binary Search Tree (Example)

- Is this a binary search tree?



Binary Search Tree (Example)

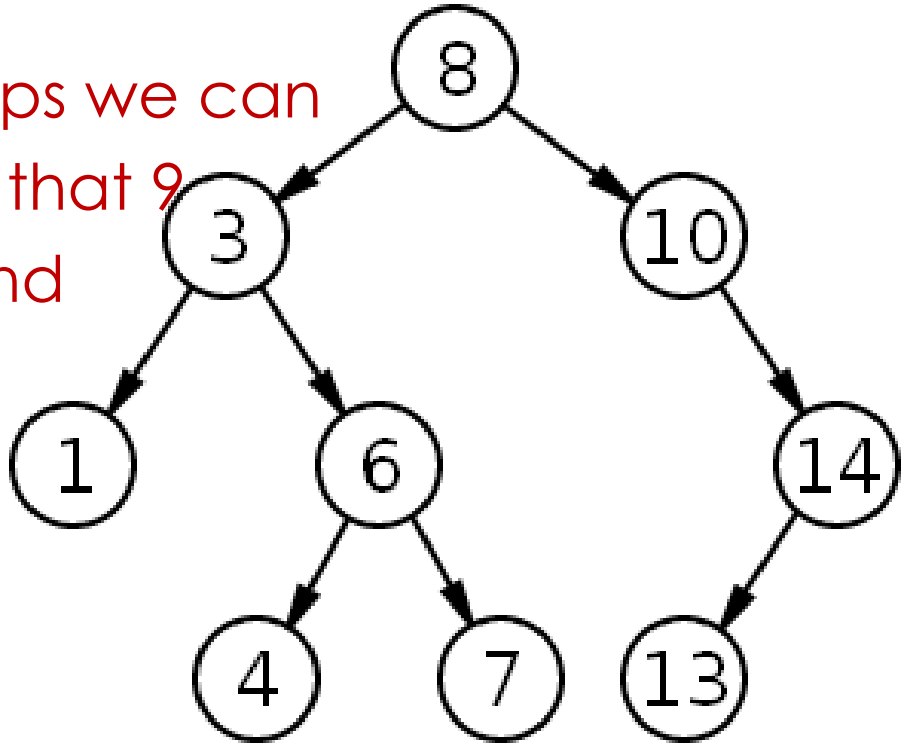
- Is this a binary search tree?



Binary Search Tree (Example)

- Find 9

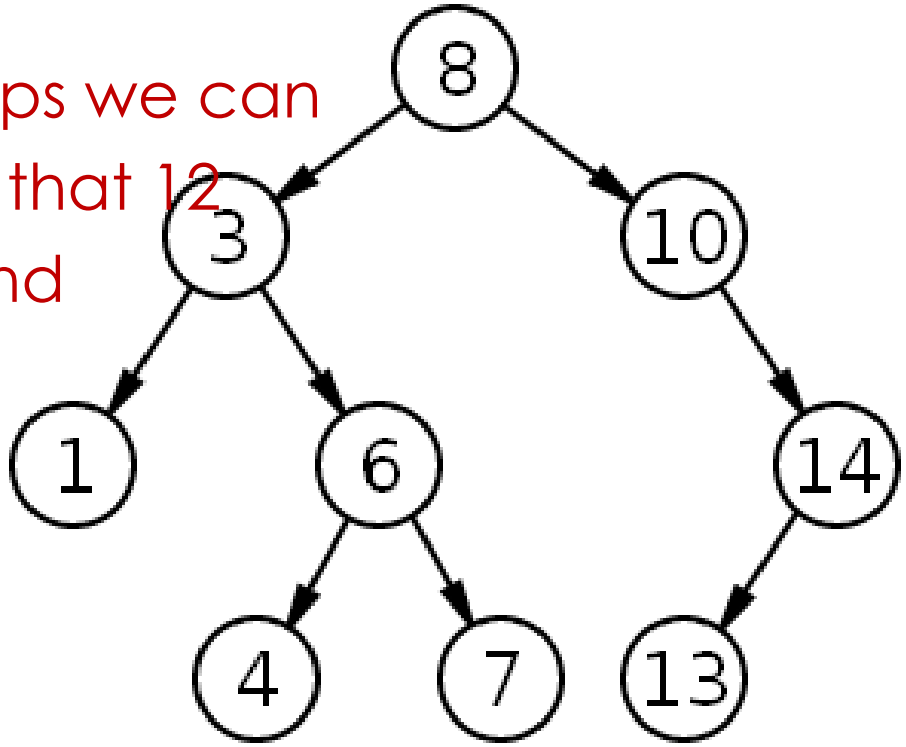
➔ After 2 steps we can conclude that 9 is not found



Binary Search Tree (Example)

- Find 12

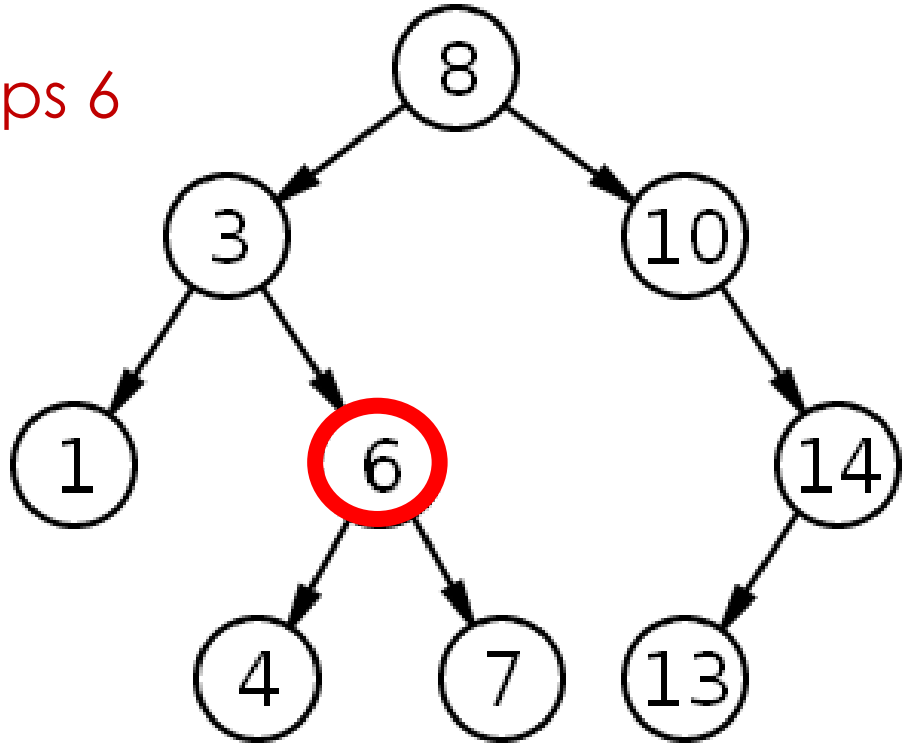
➔ After 4 steps we can conclude that 12 is not found



Binary Search Tree (Example)

- Find 6

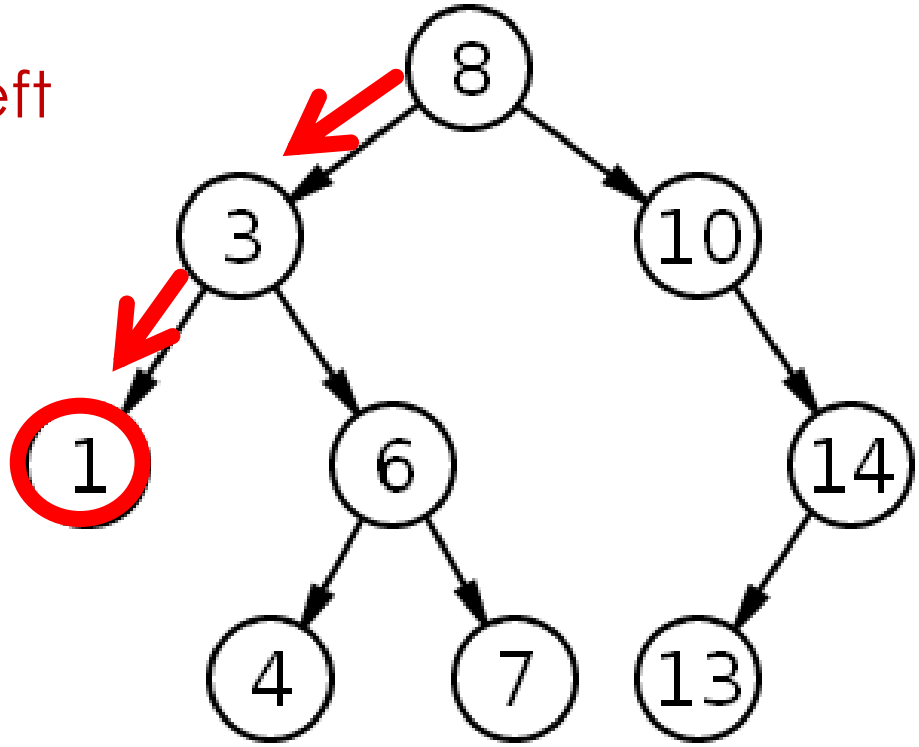
➔ After 3 steps 6 is found



Binary Search Tree (Example)

- Find the smallest value

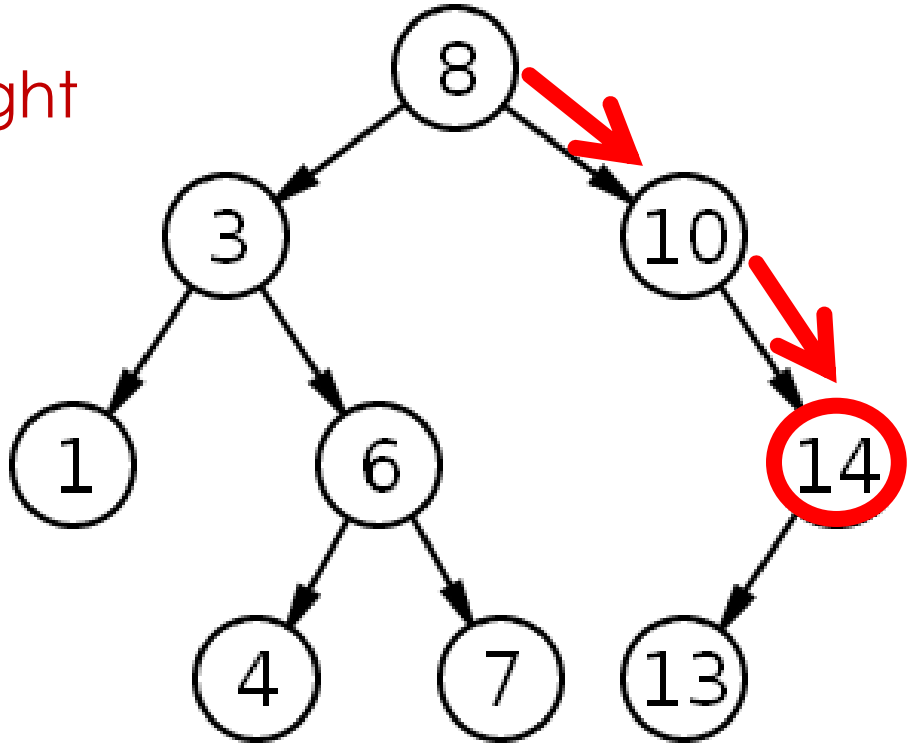
➔ At most left



Binary Search Tree (Example)

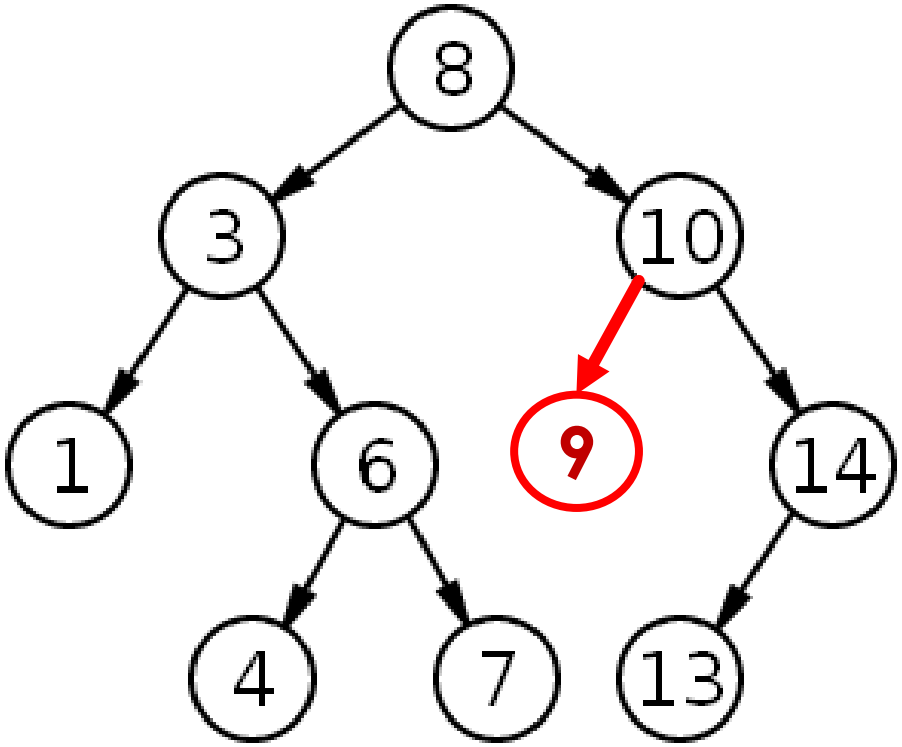
- Find the largest value

➔ At most right



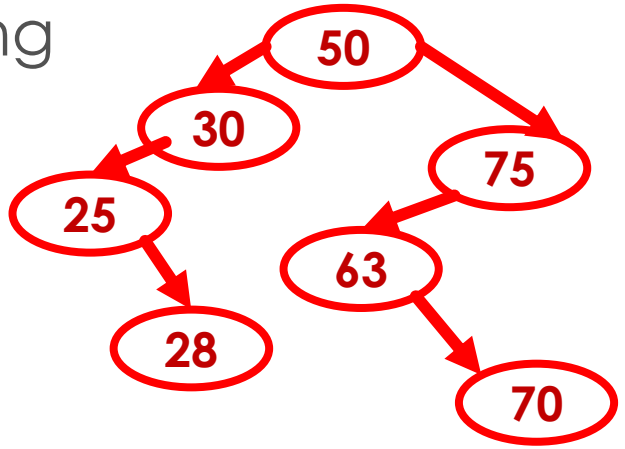
Binary Search Tree (Example)

- Insert 9



Binary Search Tree (Example)

- Draw the binary tree which would be created by inserting the following numbers in the order given:
- 50 30 25 75 63 28 70



BST Tree Implementation

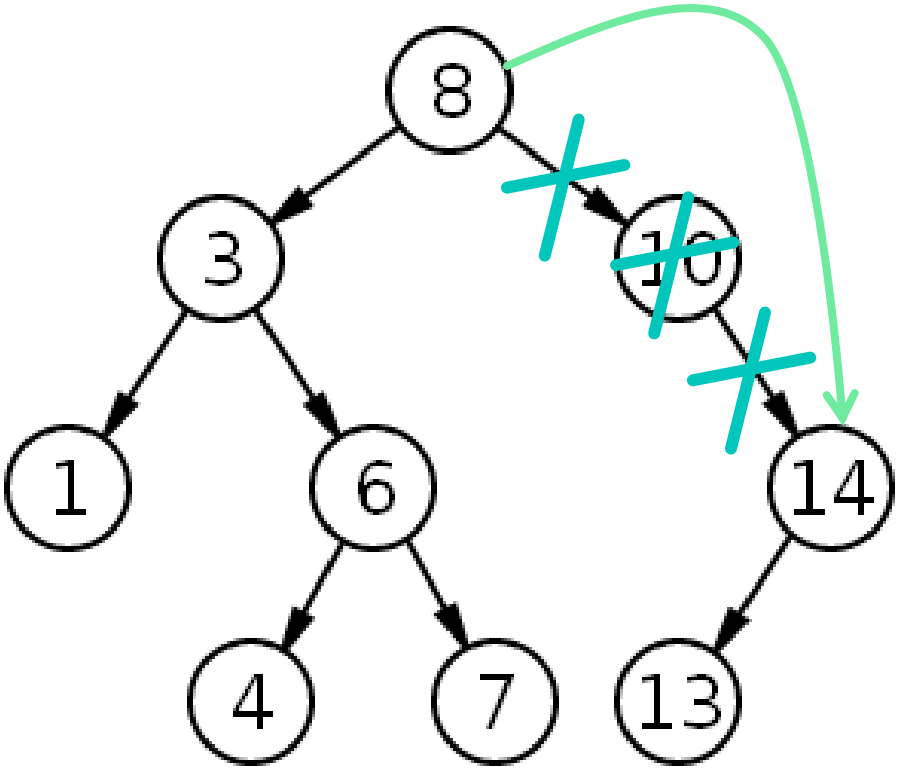
```
Void Insert(TreeType *t,EntryType item){
/*Recursion is not good here WHY??*/
 NodeType*p=(NodeType*)
  malloc(sizeof(NodeType));
 p->info=item;
 p->left=NULL;    p->right=NULL;
 if (!(*t))       *t= p;
 else{
   NodeType *pre,*cur;
      cur=*t;
    while(cur){
```

BST Tree Implementation

```
. . .
    while(cur){
          pre=cur;
          if(item<cur->info)
                cur=cur->left;
          else   cur=cur->right;
    }
    if(item <pre->info)     pre->left=p;
    else   pre->right=p;
  }
}
```

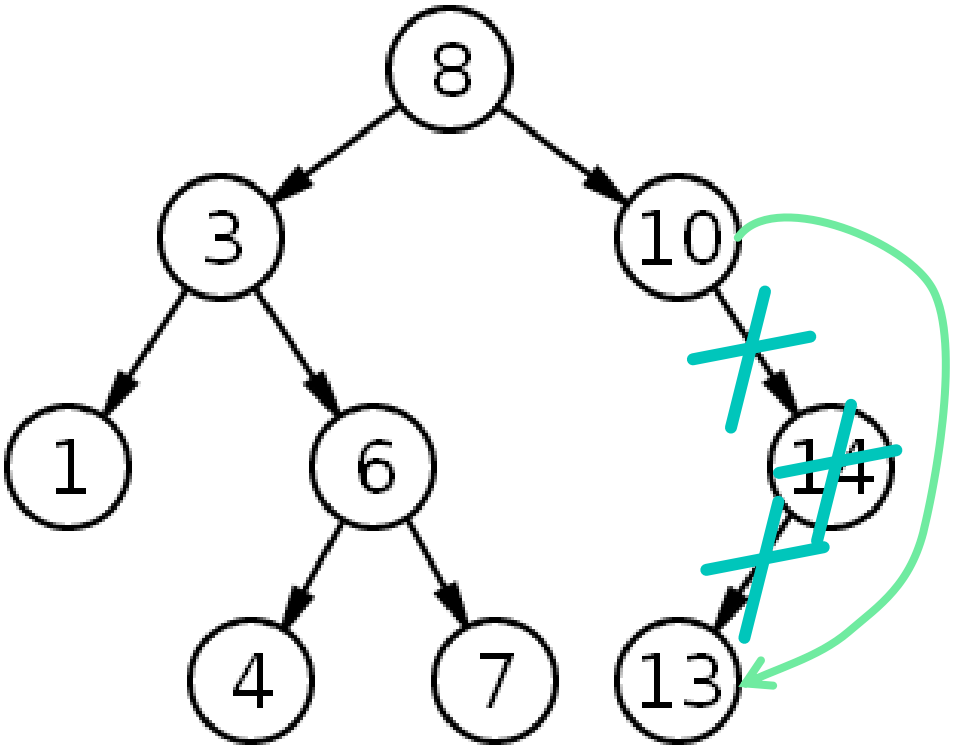
Binary Search Tree (Example)

- Delete 10



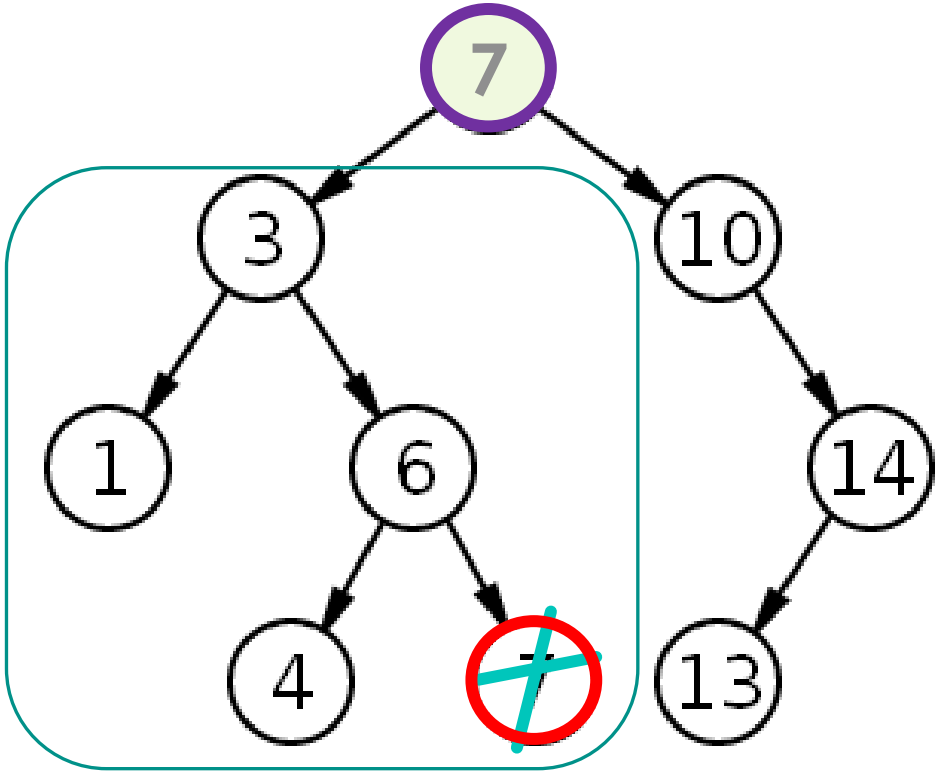
Binary Search Tree (Example)

- Delete 14



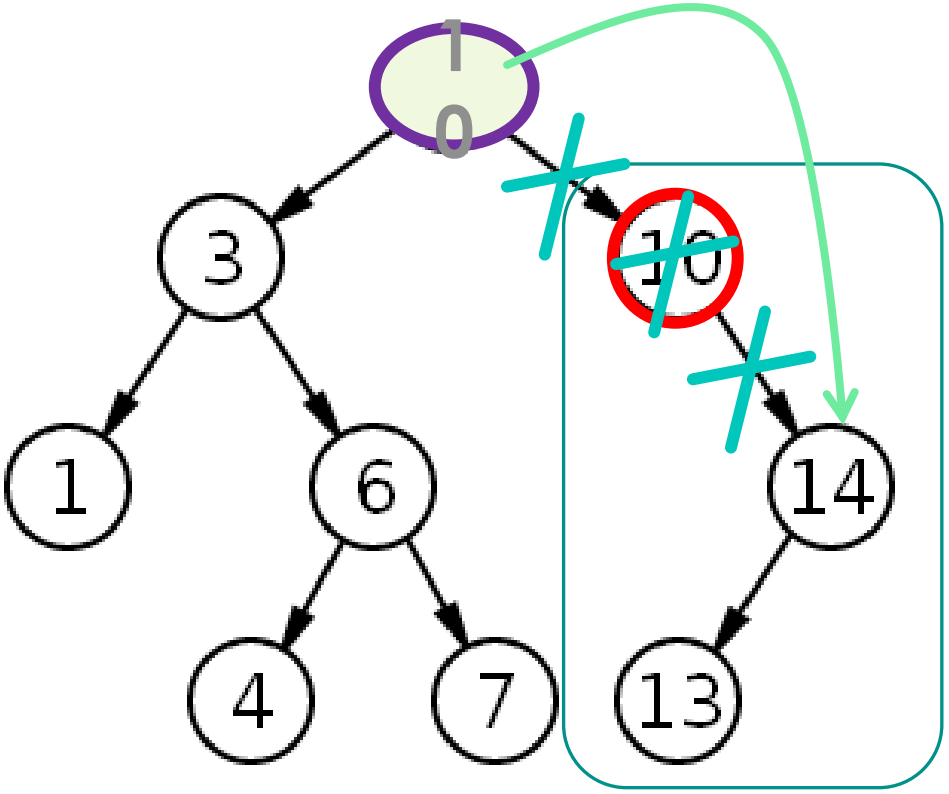
Binary Search Tree (Example)

- Delete 8



Binary Search Tree (Example)

- Delete 8



Deleting from a Binary Search Tree

- Item not present: do nothing
- Item present in leaf: remove leaf (change to null)
- Item in non-leaf with one child:

  Replace current node with that child
- Item in non-leaf with *two children*?
  - Find *largest item* in the *left subtree*
  - Remove it
  - Use it as the parent of the two subtrees
  - (Could use smallest item in right subtree)

BST Tree Implementation

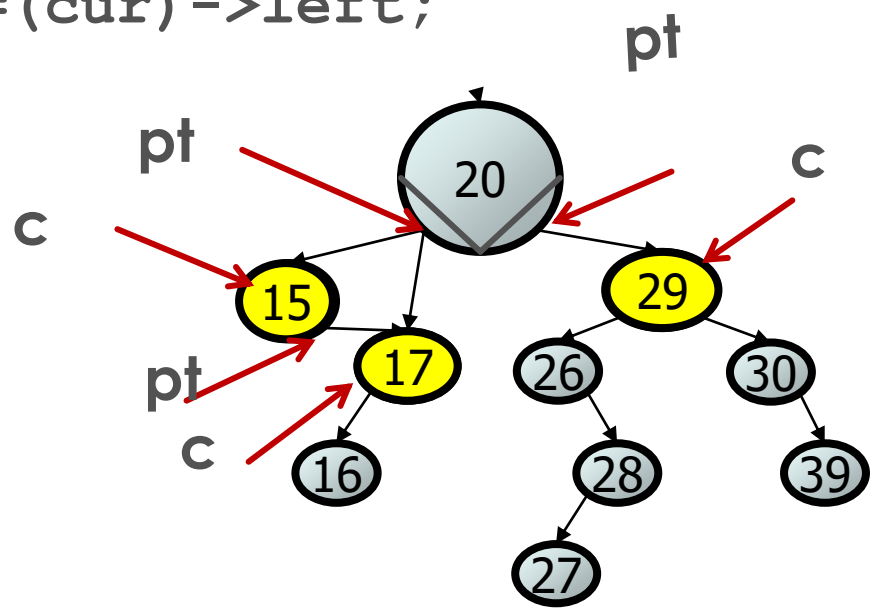
```
int Delete(TreeType *t,EntryType k){
    int found=0;    Node *cur=*t;
    Node *prev=NULL;
    while(cur && !(found=(k==cur->info))){
        prev=cur;
        if(k<cur->info)     cur=cur->left;
        else                cur=cur->right;
    }
    if (found){
```

BST Tree Implementation

```
. . .
    if (found){
        if(!prev)//Case of deleting the root
            DeleteNode(t);
        else if((k< prev->info))
                DeleteNode(&prev->left);
             else
                DeleteNode(&prev->right);
    }
    return found;
}
```

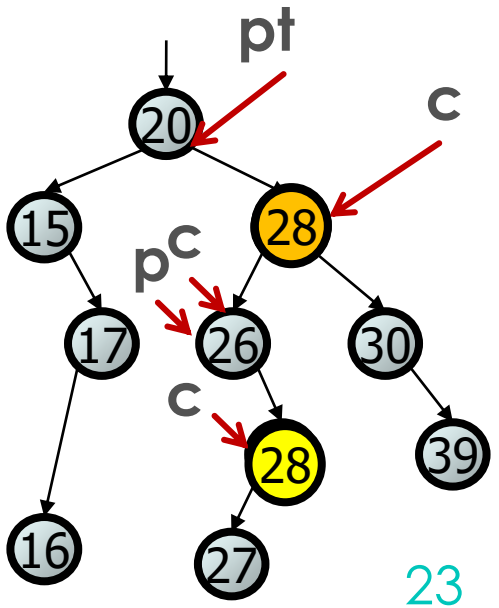
BST Tree Implementation

```
void DeleteNode(tree *pt){
    Node *cur=*pt;
    if(!(cur)->left) *pt=(cur)->right;
    else
      if(!(cur)->right) *pt=(cur)->left;
    else{




        }
    free(cur);
}
```



BST Tree Implementation

```
    . . .
    else{//third case
      cur=(cur)->left;     Node *prev=Null;
       while(cur->right){
       prev=cur;      cur=cur->right;}
       (*pt)->info=cur->info;
      if(prev)
         prev->right=cur->left;
        else
         (*pt)->left=cur->left;
       }
    free(cur);
}
```




Any
Questions