APS 105 Lecture 37-38 Notes

Last time: searching algorithms and introduced binary trees and binary search trees

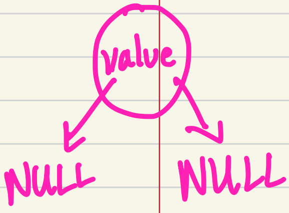
Today: Develop functions on binary search trees insert (non-recursive), search (recursive), print (recursive)

Recap:

```
typedef struct node {
        int data;
        struct node * right, left;
} Node;

typedef struct bstree{
        Node * root;
} BSTree;
```

To create a node, we can do it in a function.



```
Node* createNode (BSTree * tree, int value){
    Node * newNode = (Node*)malloc (sizeof(Node));
    if (newNode != NULL){
          newNode -> data = value;
          newNode -> right = newNode -> left = NULL;
    }
          return newNode;
}
```

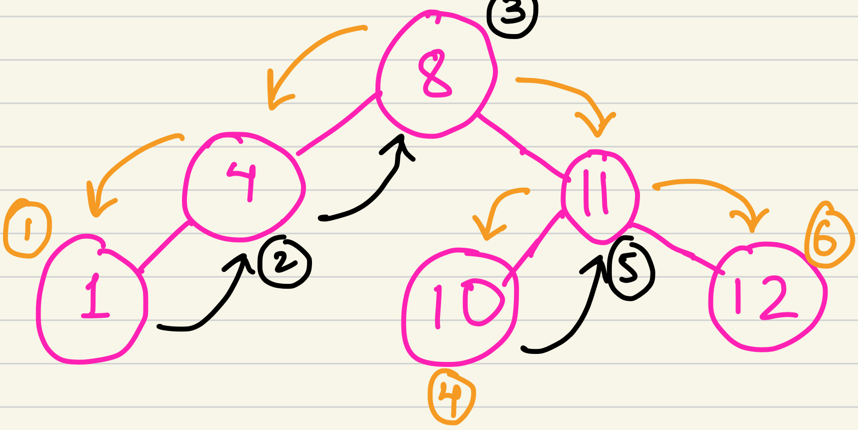
To initialize BSTree

```
int main(){
        BSTree tree;
        tree.root=NULL;
        OR
        initBSTree(&tree);
}
void initBSTree(BSTree *tree){
          tree->root=NULL;
}
```

To check if it is empty

```
bool   isEmpty(BSTree *tree){
    return(tree->root == NULL);
}
```

To print BSTree in sorted order, e.g.



This is best implement recursively, because we want to print all nodes in left subtree 1st, then root, then all nodes in right subtree.

↳ we "print" on a smaller problem!
↳ till subtree has no more nodes

BSTree *

```
void print(BSTree * tree){

    // print left subtree 1st
    print (tree -> root -> left);
```

↳ but this is of Node *

```
void printHelper (Node * n){
    if(n != NULL){
        printHelper(n -> left);
        printf("%d ", n ->data);
        printHelper ( n -> right);
    }
}
```

In-order traversal

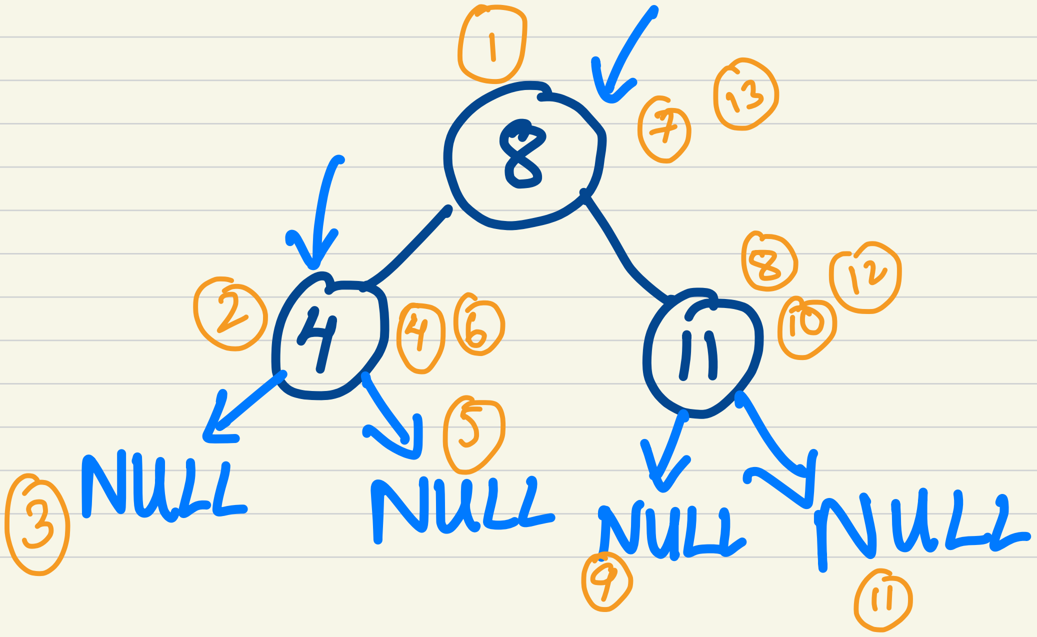
```
void print( BSTree * tree){

    printHelper (tree -> root);
```

$\underbrace{\text{tree -> root}}_{\text{Node*}}$

```
}
```

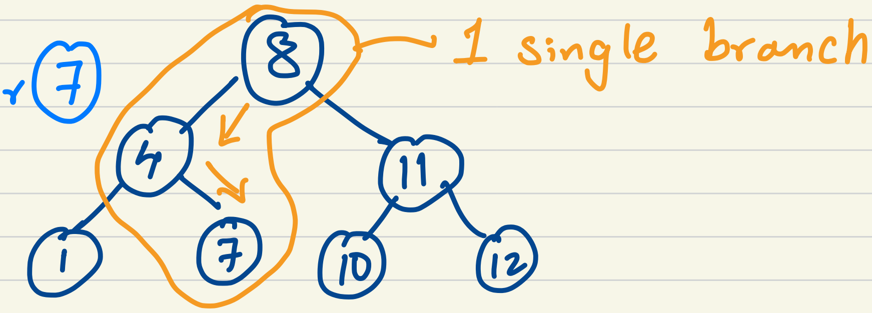
E.g.



```
printHelper( 8 )
  printHelper( 4 )
    printHelper (NULL);
    printf("%d", 4 );
    printHelper (NULL);
  printf("%d", 8 );
  printHelper ( 11 )      return to printHelper (8)
    printHelper(NULL);
    printf("%d", 11 );
    printHelper (NULL);   return to
                          printHelper (11)
```

To search for a node in the binary search tree. Easier than print because we only traverse 1 single branch not the entire tree.

E.g. Look for 7



Very easy iteratively!

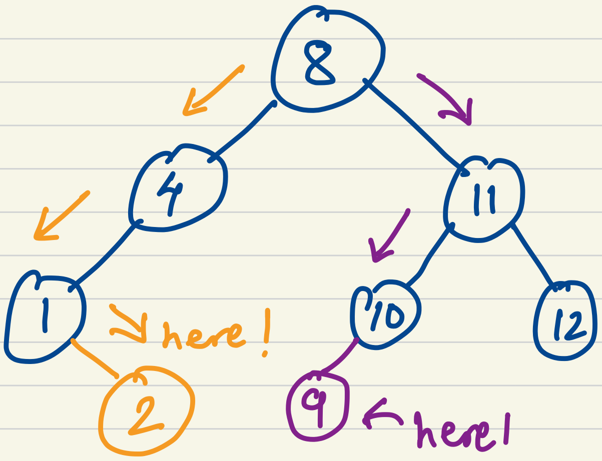
```
Node * search (BSTree * tree, int value){

    Node * current = tree -> root;
    while (current != NULL && current -> data !=
                                              value ){
        //Go left
        if( current -> data > value) {
                current = current -> left;
        }
        // Go right
        else{
               current = current -> right;
        }
    }
     //current is NULL or current -> data == value
    return current;
}
//Homework: implement search function recursively
```

To insert a node into BSTree, we need to make sure it is inserted in a place that keeps the tree order.

Insert 2

Insert 9



```
bool insert(BSTree * tree, int value){
        Node * current= tree->root, *parent = NULL;
If tree  { if(tree -> root == NULL){
is                tree -> root = createNode(value);
empty             return tree -> root != NULL;
           }
        while ( current != NULL){
              parent= current;
Go left  -> if(current -> data > value){
                   current = current -> left;
            }
Go right -> else{
                   current = current -> right;
            }
        } // current is NULL, current lost access to tree
        if (value < parent-> data){
Return if we were  parent -> left = createNode(value);
able to insert     return parent -> left != NULL;
        }
        else {
               parent ->right= createNode(value);
               return parent->right != NULL;
        }
}
```