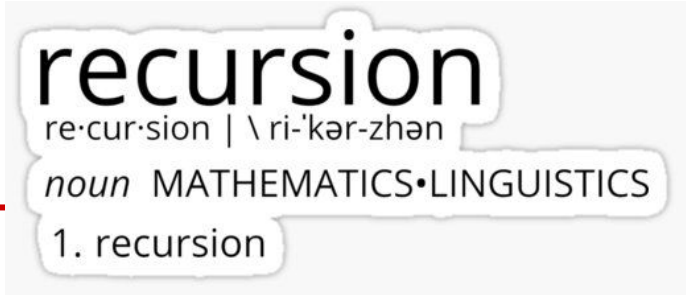
Session 7: Binary Search Trees

Data Structures and Algorithm 1 - Lab

Yahya Tawil

19 Nov 2021

Recursion Basics

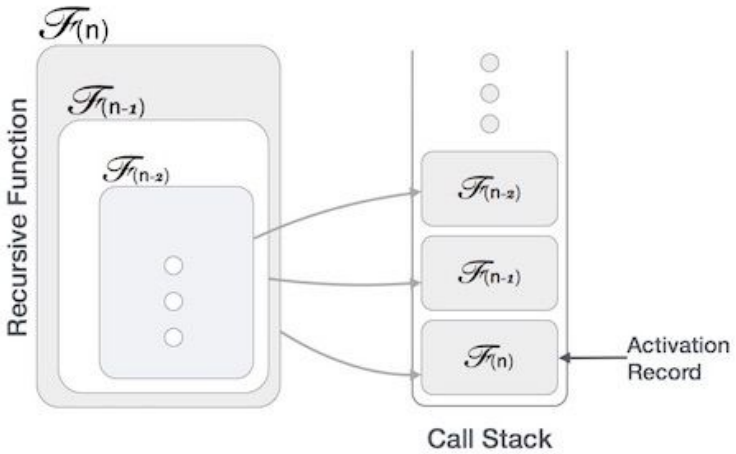


- Some computer programming languages allow a module or function to call itself. This technique is known as **recursion**.

- Properties:
  - **Base criteria** − There must be at least one base criteria or condition, such that, when this condition is met the function stops calling itself recursively.
  - **Progressive approach** − The recursive calls should progress in such a way that each time a recursive call is made it comes closer to the base criteria.

Recursion Implementation

- Many programming languages implement recursion by means of **stacks.**
- Generally, whenever a function (caller) calls another function (callee) or itself as callee, the caller function transfers execution control to the callee.
- The caller function needs to start exactly from the point of execution where it puts itself on hold.



https://www.tutorialspoint.com/data_structures_algorithms/recursion_basics.htm

Recursion Real Life Example

Suppose you are standing in a long queue of people. How many people are directly behind you in the line?

Rules	One person can see only the person standing directly in front and behind. So, one can't just look back and count.	Each person is allowed to ask questions from the person standing in front or behind. How can we solve this problem recursively?
Recursive solution	You look behind and sees if there is a person there. If not, then you can return the answer "0". If there is a person, repeat step 1, and wait for a response from the person standing behind.	Once a person receives a response, they add 1 for the person behind them, and they respond to the person that asked them or the person standing in front of them.

https://www.enjoyalgorithms.com/blog/recursion-explained-how-recursion-works-in-programming

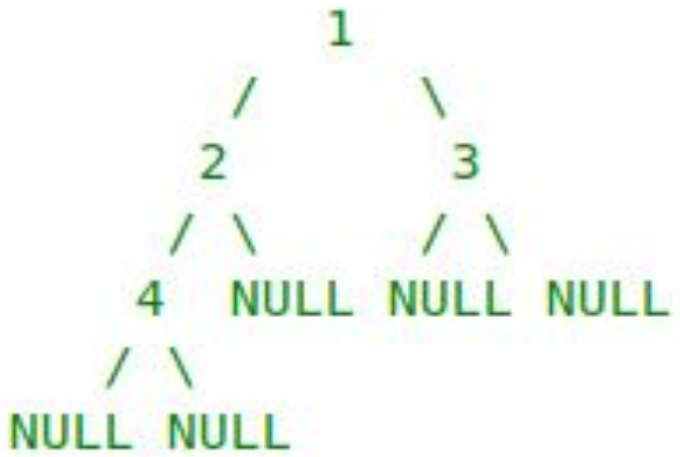
Recursion Real Life Example

```
int peopleCount(Person curr)
{
  if (noOneBehind(currPerson))
  {
        return 0
  }
  else
  {
        Person personBehind = curr.getBehind()
        return peopleCount(personBehind) + 1
  }
}
```

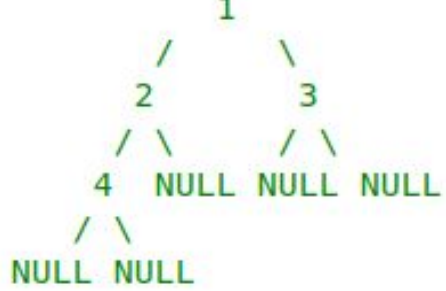
https://www.enjoyalgorithms.com/blog/recursion-explained-how-recursion-works-in-programming

Recersuive Tree Depth Calculation

```
int maxDepth(Node* node)
{
    if (node == NULL) return 0;
    else
    {
        int lDepth = maxDepth(node->left);
        int rDepth = maxDepth(node->right);
        if (lDepth > rDepth)
            return(lDepth + 1);
        else return(rDepth + 1);}}
```



Recersuive Tree Depth Calculation




```
int maxDepth(Node* node)
{
    if (node == NULL) return 0;
    else
    {
        int lDepth = maxDepth(node->left);
        int rDepth = maxDepth(node->right);
        if (lDepth > rDepth)
            return(lDepth + 1);
        else return(rDepth + 1);}}
```

Stack

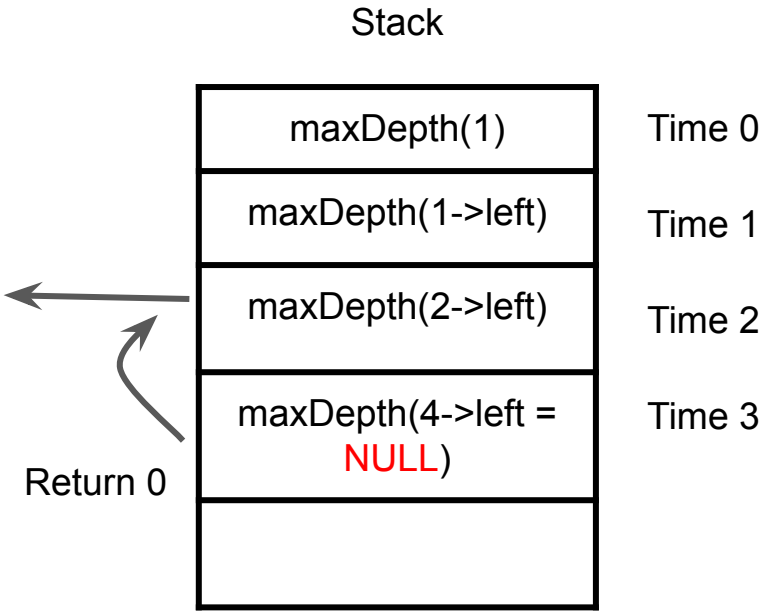
maxDepth(1)	Time 0
maxDepth(1->left)	Time 1
maxDepth(2->left)	Time 2
maxDepth(4->left)	Time 3

Recersuive Tree Depth Calculation

```
int maxDepth(Node* node)
{
    if (node == NULL) return 0;
    else
    {
        int lDepth = maxDepth(node->left);
        int rDepth = maxDepth(node->right);
        if (lDepth > rDepth)
            return(lDepth + 1);
        else return(rDepth + 1);}}
```

0

(1) Return 0



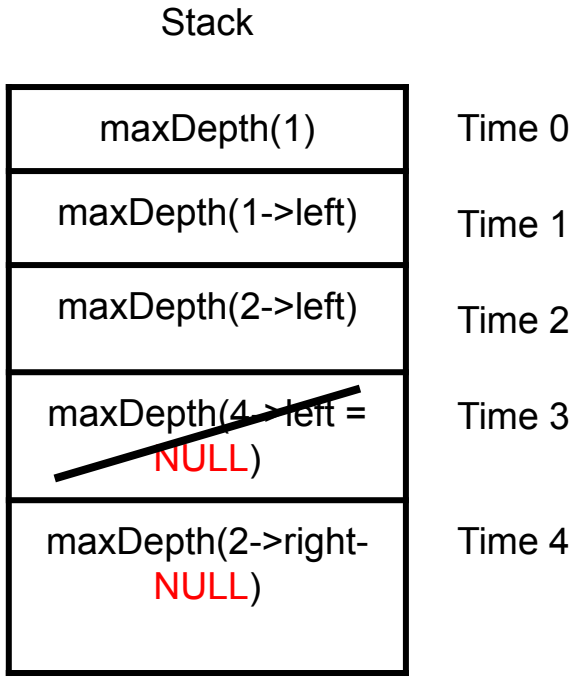
Recersuive Tree Depth Calculation

```
int maxDepth(Node* node)
{
    if (node == NULL) return 0;
    else
    {
        int lDepth = maxDepth(node->left);
        int rDepth = maxDepth(node->right);
        if (lDepth > rDepth)
            return(lDepth + 1);
        else return(rDepth + 1);}}
```

0

0

(1) Return 0

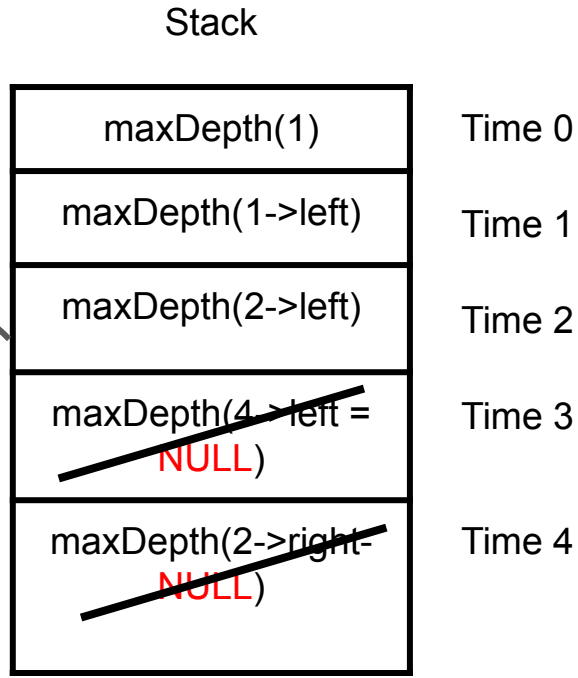


Recersuive Tree Depth Calculation

```
int maxDepth(Node* node)
{
    if (node == NULL) return 0;
    else
    {
        int lDepth = maxDepth(node->left);
        int rDepth = maxDepth(node->right);
        if (lDepth > rDepth)
            return(lDepth + 1);
        else return(rDepth + 1);}}
```

1

(1) Return 1



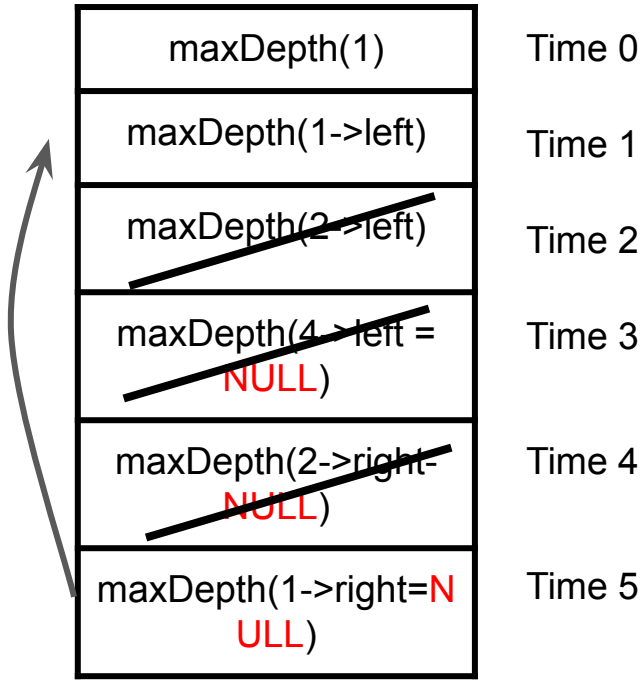
Recersuive Tree Depth Calculation

```
int maxDepth(Node* node)
{
    if (node == NULL) return 0;
    else
    {
        int lDepth = maxDepth(node->left);
        int rDepth = maxDepth(node->right);
        if (lDepth > rDepth)
            return(lDepth + 1);
        else return(rDepth + 1);}}
```

1

0

Stack



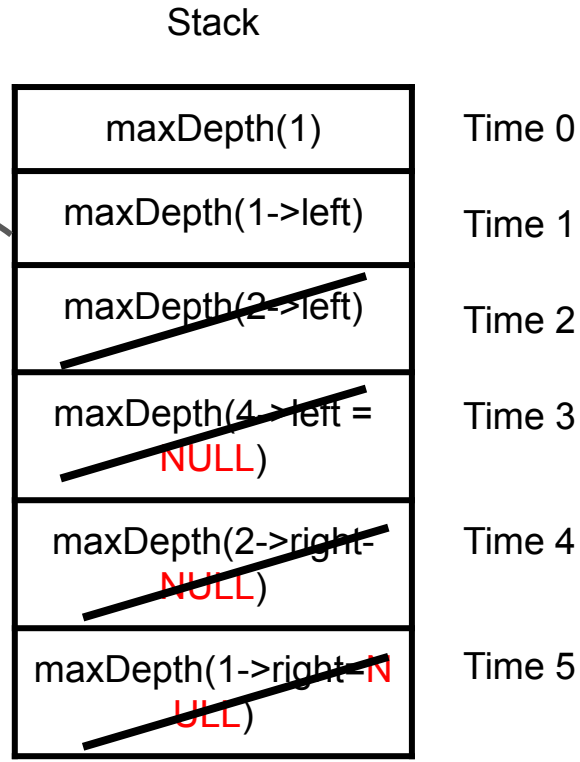
(1) Return 0

Recersuive Tree Depth Calculation

```
int maxDepth(Node* node)
{
    if (node == NULL) return 0;
    else
    {
        int lDepth = maxDepth(node->left);
        int rDepth = maxDepth(node->right);
        if (lDepth > rDepth)
            return(lDepth + 1);
        else return(rDepth + 1);}}
```

2

(1) Return 2

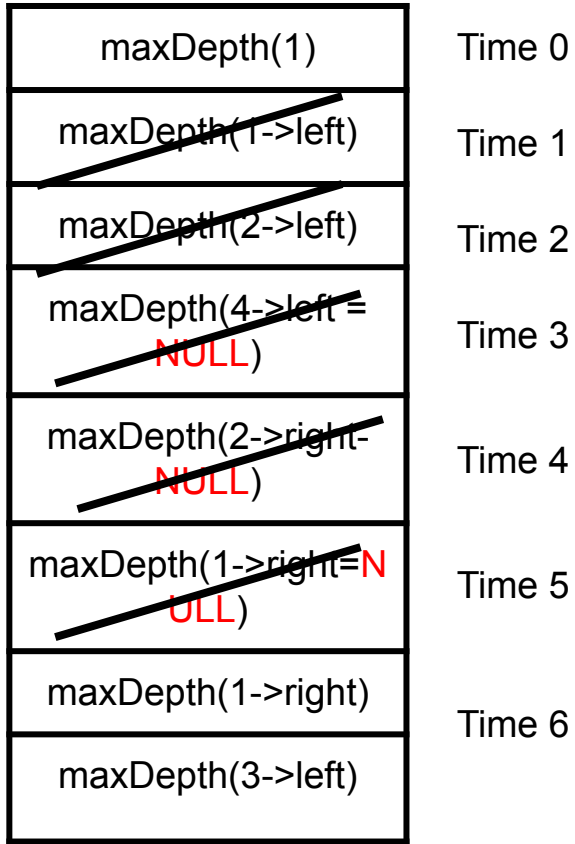


Recersuive Tree Depth Calculation

```
int maxDepth(Node* node)
{
    if (node == NULL) return 0;
    else
    {
        int lDepth = maxDepth(node->left);
        int rDepth = maxDepth(node->right);
        if (lDepth > rDepth)
            return(lDepth + 1);
        else return(rDepth + 1);}}
```

0

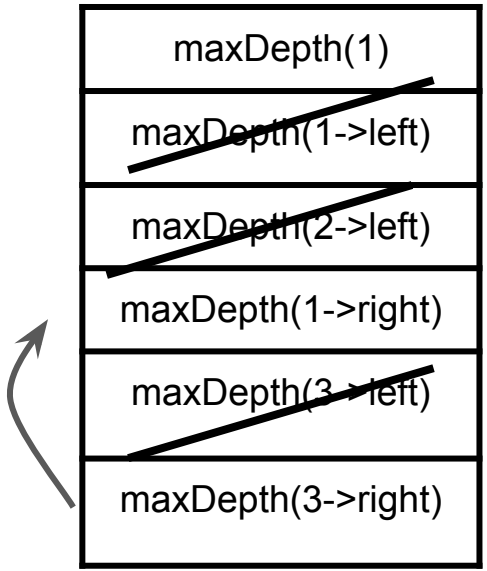
(1) Return 0



Recersuive Tree Depth Calculation

```
int maxDepth(Node* node)
{
    if (node == NULL) return 0;
    else
    {
        int lDepth = maxDepth(node->left);
        int rDepth = maxDepth(node->right);
        if (lDepth > rDepth)
            return(lDepth + 1);
        else return(rDepth + 1);}}
```

0 0 (1) Return 0



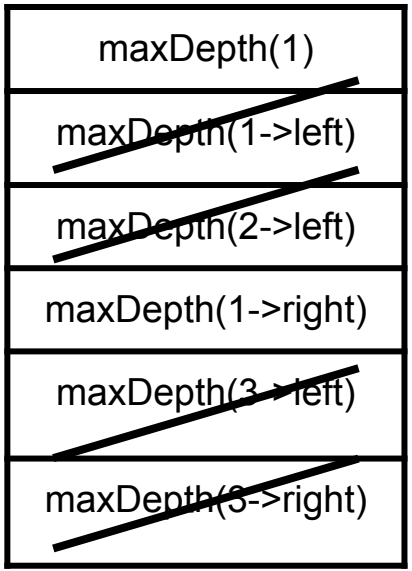
Recersuive Tree Depth Calculation

```
int maxDepth(Node* node)
{
    if (node == NULL) return 0;
    else
    {
        int lDepth = maxDepth(node->left);
        int rDepth = maxDepth(node->right);
        if (lDepth > rDepth)
            return(lDepth + 1);
        else return(rDepth + 1);}}
```

0

0

(1) Return 1

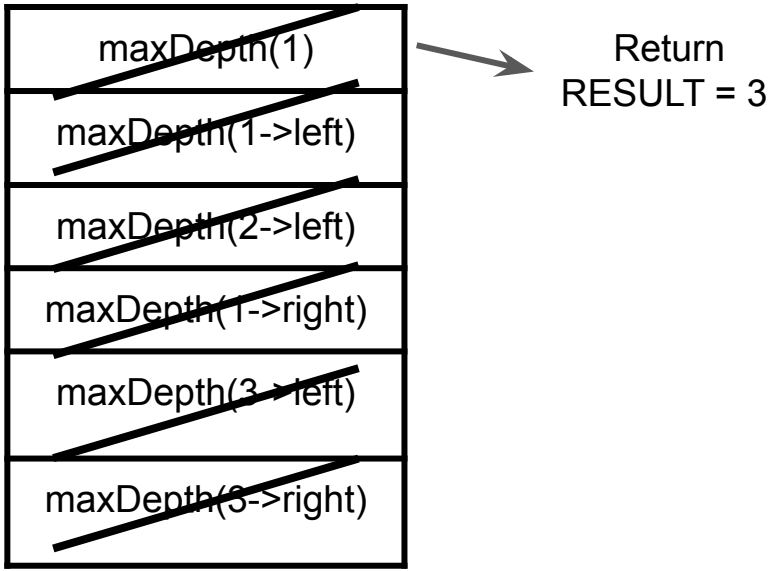


Recersuive Tree Depth Calculation

```
int maxDepth(Node* node)
{
    if (node == NULL) return 0;
    else
    {
        int lDepth = maxDepth(node->left);
        int rDepth = maxDepth(node->right);
        if (lDepth > rDepth)
            return(lDepth + 1);
        else return(rDepth + 1);}}
```

2

1



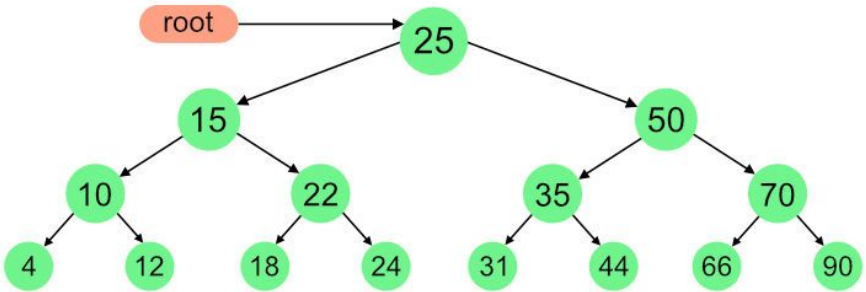
Binary Tree Traversal Example: Postorder

```
void printPostorder(struct Node*
node)
{
    if (node == NULL)
        return;

    // first recur on left subtree
    printPostorder(node->left);

    // then recur on right subtree
    printPostorder(node->right);

    // now deal with the node
    cout << node->data << " ";
}
```



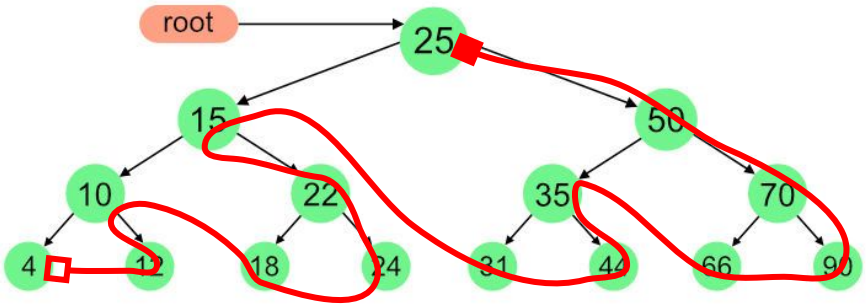
Binary Tree Traversal Example: Postorder

```
void printPostorder(struct Node*
node)
{
    if (node == NULL)
        return;

    // first recur on left subtree
    printPostorder(node->left);

    // then recur on right subtree
    printPostorder(node->right);

    // now deal with the node
    cout << node->data << " ";
}
```



Solution 4,12,10,18,24,22,15,31,44,35,66,90,70,50,25

Remark: it ends with the root

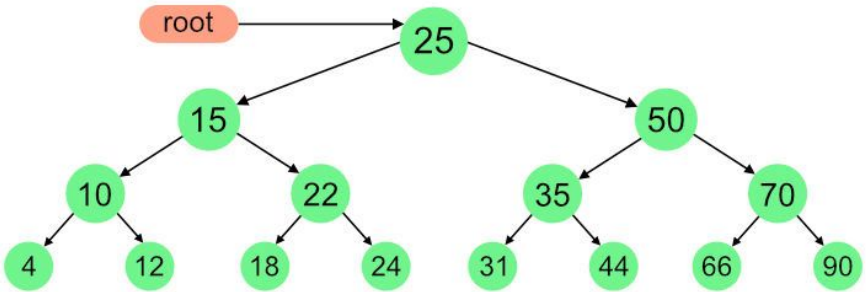
Binary Tree Traversal Example: Inorder

```
void printInorder(struct Node* node)
{
    if (node == NULL)
        return;

    /* first recur on left child */
    printInorder(node->left);

    /* then print the data of node */
    cout << node->data << " ";

    /* now recur on right child */
    printInorder(node->right);
}
```



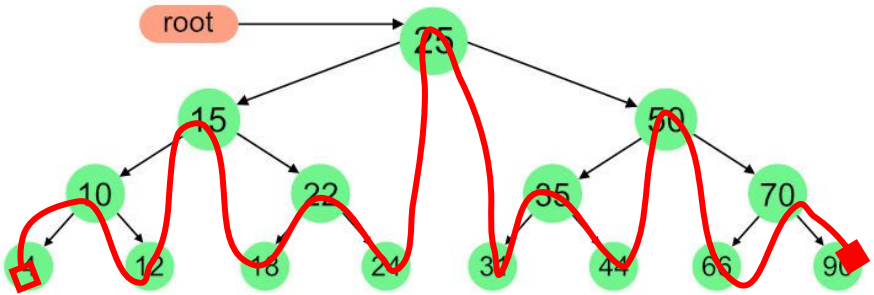
Binary Tree Traversal Example: Inorder

```
void printInorder(struct Node* node)
{
    if (node == NULL)
        return;

    /* first recur on left child */
    printInorder(node->left);

    /* then print the data of node */
    cout << node->data << " ";

    /* now recur on right child */
    printInorder(node->right);
}
```



Solution 4,12,10,15,18,22,24,25,31,35,44,50,66,70,90

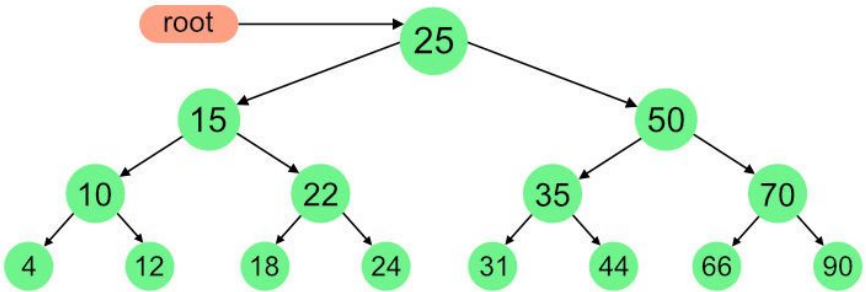
Binary Tree Traversal Example: Preorder

```
void printPreorder(struct Node*
node)
{
    if (node == NULL)
        return;

    /* first print data of node */
    cout << node->data << " ";

    /* then recur on left subtree */
    printPreorder(node->left);

    /* now recur on right subtree */
    printPreorder(node->right);
}
```



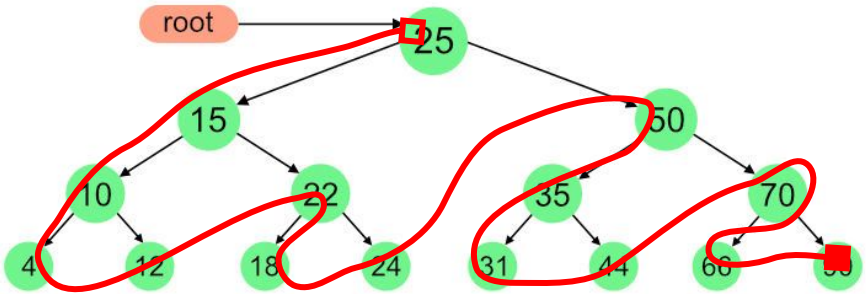
Binary Tree Traversal Example: Preorder

```
void printPreorder(struct Node*
node)
{
    if (node == NULL)
        return;

    /* first print data of node */
    cout << node->data << " ";

    /* then recur on left subtree */
    printPreorder(node->left);

    /* now recur on right subtree */
    printPreorder(node->right);
}
```



Solution 25,15,10,4,12,22,18,24,50,35,31,44,70,66,90

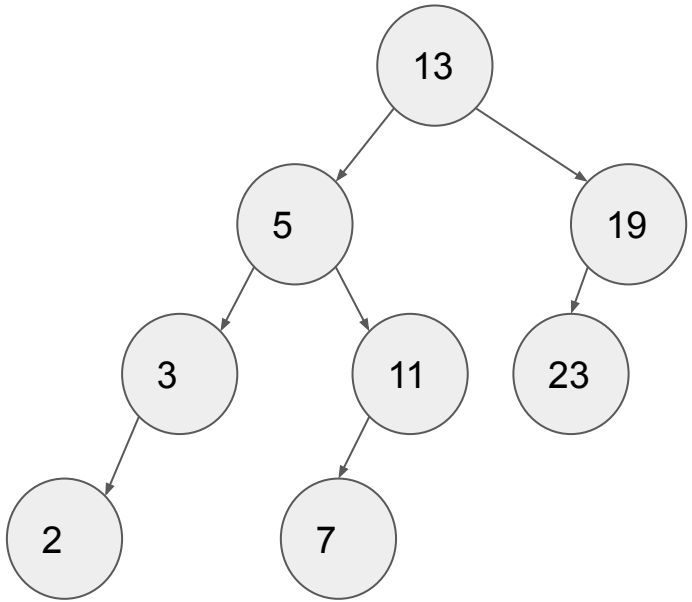
Remark: it starts with the root

Binary Tree Traversal Example

Given a BST in **pre-order** as {13,5,3,2,11,7,19,23}, draw this BST and determine if this BST is the same as one described in **post-order** as {2,3,5,7,11,23,19,13}

Binary Tree Traversal Example

Given a BST in **pre-order** as {13,5,3,2,11,7,19,23}, draw this BST and determine if this BST is the same as one described in **post-order** as {2,3,5,7,11,23,19,13}



Assignment 7: Exercise 1

Given a BST in **pre-order** as {20 16 5 18 17 19 60 85 70}, draw this BST and determine if this BST is the same as one described in **In-order** as {5 16 17 18 19 20 60 70 85}

Lab Project

Write a program do the following:

- Takes a Goodreads dataset and extract author names and book titles. (20%)
- Build a BST for authors and another BST for books title. (30%)
- Fetch a list of all books that start with the query string .i.e (an introduction) to fetch all books start with this string. (20%)
- The same thing while searching for authors name. (20%)

Note: An extra marks will be assigned for good code design, documenting the code through comments, and making a simple GUI. (10%)

Dataset Link: https://www.kaggle.com/jealousleopard/goodreadsbooks

Lab Project

```
FILE *fp;
char str1[10], str2[10], str3[10], str4[10];

fp = fopen("test.csv", "r");

if(NULL == fp)
{
        printf("\nError in opening file.");
        return 0;
}
while(EOF != fscanf(fp, " %[^,], %[^,], %[^,], %s, %s, %s, %s ", str1, str2, str3, str4)) // d1,d2,d3,d4
{
        printf("\n%s %s %s %s", str1, str2, str3, str4);
}
fclose(fp);
```

Lab Quiz (10 Dec)

- 30 Min.
- Multiple choice questions.
- No need to memorize the codes. Only understand it.
- Date: 10 Dec
- Mark: 20% of total lab marks.