CSCI 2275 – Programming and Data structures
Instructor: Hoenigman
Recitation 7

# Recursions and Trees

**Objectives:**

1. What is Recursion and How is it useful
2. Trees
3. Binary Trees and BST
4. Tree Traversal
5. Exercise

## Recursion

Recursion is when a function basically calls itself to solve the problem. Now as we know that the function definition is the same how would calling itself solve anything? Also, when does it stop? Why are we even using recursion? Why is it even a thing?

Before we dive deep into it lets just get few things straight.

1. When a function is calling itself (recursive call) it is always (mostly) with different parameters.
2. All functions with recursive call will have base cases. Without a base case recursion is pointless and will break your code.
3. Recursion is used when the given problem can be broken into smaller problems and those smaller problems can be further broken down and everything comes down to the base cases

Now that we have seen this lets look at a simple example for recursion.

Consider you want to find factorial of a number n.

Now n! = n * n-1 * n-2 * n-3 ……. * 1

We can rewrite this as
n! = n * (n-1)!

We can write (n-1)! = (n-1)* (n-2)! And so on

We can see that this everything will in the end be a "base case" of 1! = 1* 0! and we know that 0! = 1

So lets see how a recursive function look like:

```
int factorial (int n){
        //Base case
        if(n==0){
                return 1;
        }
        return n*factorial(n-1);
}
```

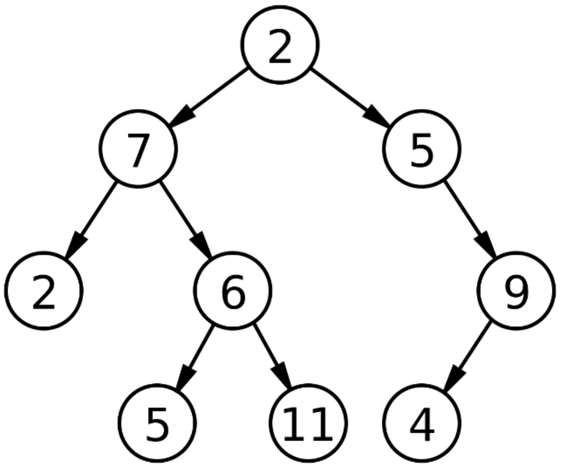
## Trees

Rules of Trees:

1. All non-empty trees will have a root node
2. All subset of Trees – Child nodes of trees can be their own smaller trees
3. Leaf nodes are the ones which have no children
4. All nodes will have at the most 1 node pointing to it (parent node). Root will not have a parent

**Binary Trees**

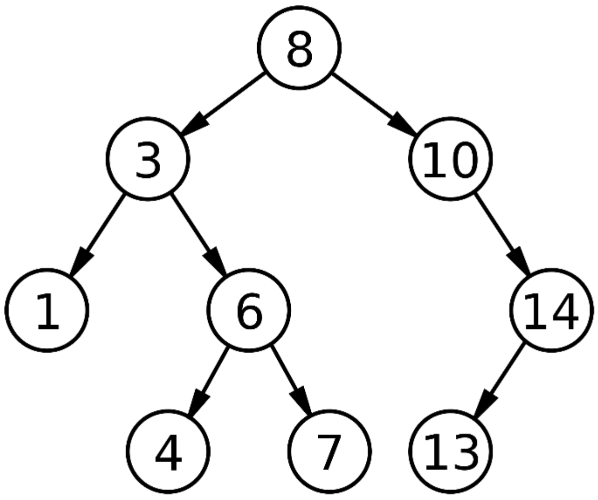
All the rules of Trees apply with one more rule:
A node will have at the most 2 nodes as children – left and right
(These are logical direction)



**Binary Search Trees**

A binary search tree is a binary tree (refer to figure 2). It may be empty. If not empty, then it satisfies the following properties:

1. Each node has exactly one key and the keys in the tree are distinct.
2. The keys in the left subtree are smaller than the key in the root.
3. The keys in the right subtree are larger than the key in the root.
4. The left and right subtrees are also binary search trees.



**Insert into the Binary Search tree**

Remember the property of binary tree. Every value in the left subtree is smaller than the value at root and every value at right subtree is larger than the value at the root. So, when we are inserting a new node in the tree we should maintain that property. Hence, we should first compare the value of to be inserted with the current node. If the value is smaller than the value at the current node, move on to left subtree. If value is greater move on to the right subtree. If there is no left or right subtree we will insert the node there.

**Search in the tree**

Remember searching in an array or linked list. We need to traverse the whole array to check presence of an element. Hence it may take O(n) time. But using the binary search tree's property searching for a key in a binary search tree can be done in O(logn) time. We will use a similar algorithm to what we used in insert. We go left if we are looking for smaller value, right when we are looking for bigger value. If the left or right does not exist, we can say that it does not exist.

**Traversal of a Binary Tree**

There will be many situations when you may need to traverse the tree. For example, you may like to print out the node values of the tree. There are many ways to traversing a tree. Here we will discuss three of them.

1. In order traversal – Left – Root – Root
2. Pre order traversal – Root – Left – Right
3. Post order traversal – Left – Right- Root

**Exercise**

Implement the insert, search and In order traversal of a Binary search tree. Then do the following:

1. Insert 5
2. Insert 2
3. Insert 7
4. Insert 4
5. Insert 6
6. Search 2
7. Insert 1
8. Print the in order traversal