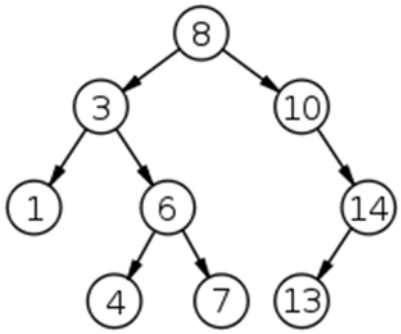
CS 2420 Program 2 – 24 points
Spring 2018

Fun With Recursion

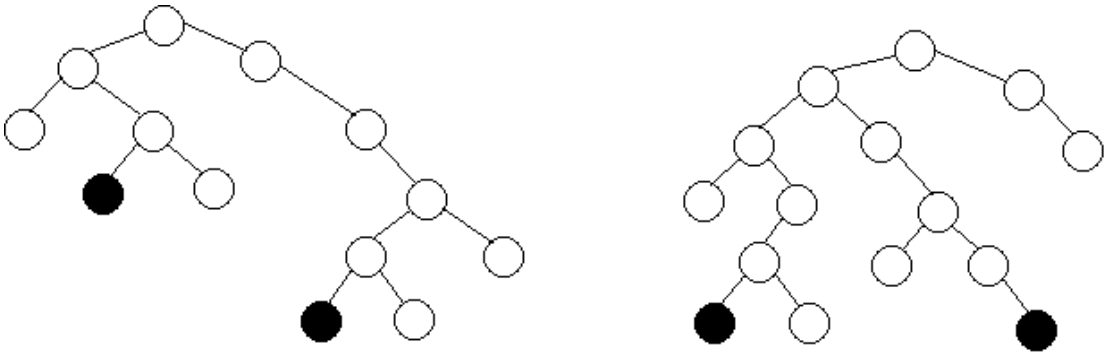
Objective: Fun with recursion. You have been given starter code which creates and prints binary search trees. You will need to modify the code. You are to write methods to accomplish the following:

1. (1 points) Write a function, **printShort**, that prints a message, and prints the keys (using an inorder traversal) of a tree, given the root.
2. (2 points) Write a function **(fall)** that removes all of the leaf nodes of a *binary search tree* rooted at root. Assume the binary search tree stores integers. A leaf node has no children.
3. (2 points) Write a function **(successor)** to find the inorder successor of a node. You should use the parent link. This function can be written in less than 10 lines of code – but that's only a guideline.
4. (2 points) Write the function ***closestCommon*** which accepts two node values in a binary tree and determines the closest ancestor to both. *It returns a pointer to the ancestor, if one exists, or null otherwise. For example in the tree below, the closestCommon(3,13) is 8 and the closestCommon(1,7) is 3. The closestCommon(3,11) is NULL. Finding the nodes in question will require a brute force search (since the tree is not known to have any specific order).*



5. (3 points) By definition, the *width* of a tree is the length of the longest path between two leaves in the tree (not considering the direction of the arcs). The diagram below shows two trees each with width eight, the leaves that form the ends of a longest path are shaded.

   It can be shown that the width of a tree *T* is the largest of the following quantities:
   - The width of *T*'s left subtree
   - The width of *T*'s right subtree
   - The longest path between leaves that goes through the root of *T* (this can be computed from the heights of the subtrees of *T*). Both of these trees have width 8.

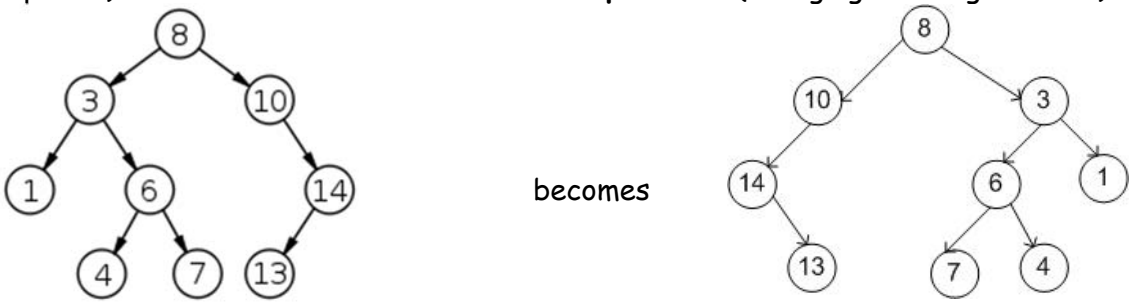


Here's code that's almost a direct translation of the three properties above (assuming the existence of a standard O(1) max function that returns the larger of two values).

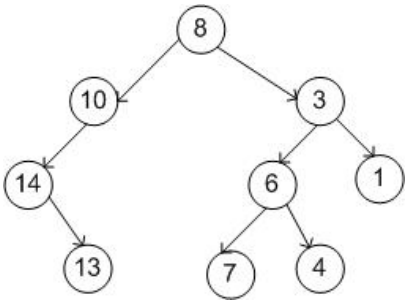
```
// returns width of tree rooted at  t
int width (TreeNode * t)
{
    if (t == NULL)  return 0;
    int leftW = width (t->left);
    int rightW = width(t->right);
    int rootW  = height(t->left) + height(t->right)+2;

    return max(rootW, max(leftW, rightW));
}
```

However, the function as written does not run in *O(n)* time because each subtree is searched twice. Write a more efficient version which computes width and height in a single traversal.

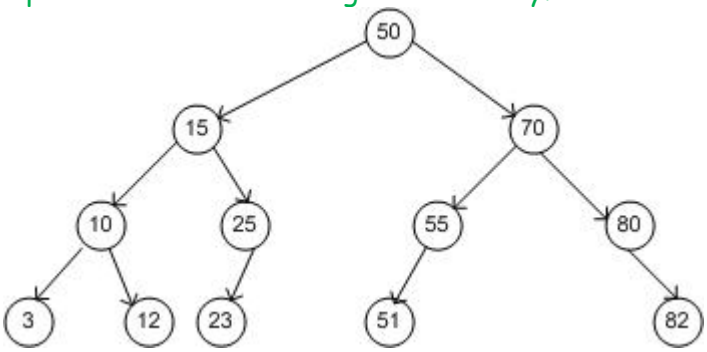
6. (2 points) Write a recursive method which **flips** a tree (changing the original tree)



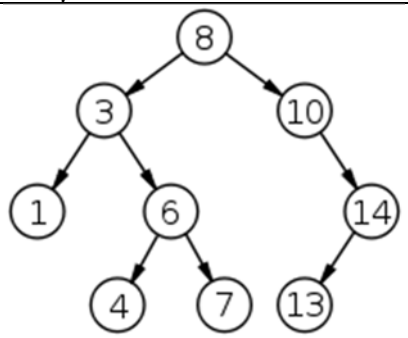
7. (3 points) Write a routine **buildFromPreorder** which does the following: Given the preorder traversal of a tree reconstruct the tree. The input consists of the value to be inserted followed by the number of children. If there is a single child, it can be a left or right child. For example, given the order: 8 2 10 1 14 1 13 0 3 2 6 2 7 0 4 0 1 0 , you would create the following tree. Make sure parent pointers have been assigned correctly.



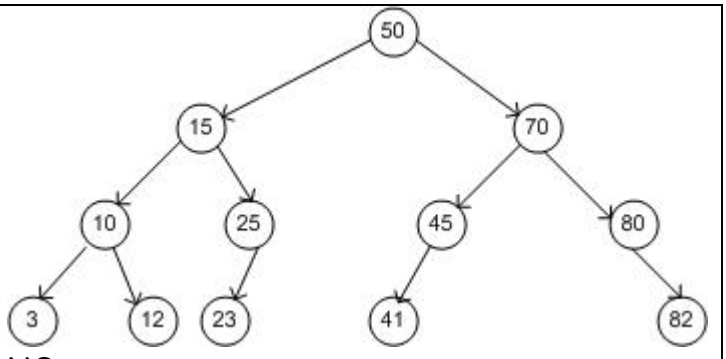
8. (3 points) **Perfect balance.** Given a set of keys, create a binary search tree of perfect balance (the tree is of minimum height). You can assume the values are given in order. Given the keys,3,10,12,15,23,25,50,51,55,70,80,82, you might produce the following tree. Make sure parent pointers have been assigned correctly.



9. (3 points) Write a recursive function (isBST) to determine whether a binary tree is a binary search tree.

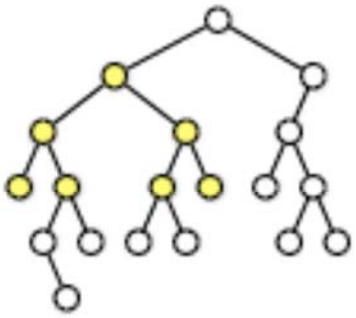


YES



NO

10. (3 points) **getMaxCompleteSubtree** For this problem, a subtree of a binary tree means any connected subgraph; a binary tree is complete if every leaf has exactly the same depth. Write a recursive algorithm to compute the largest complete subtree of a given binary tree. Your algorithm (getMaxCompleteSubtree) should find the root of the best complete tree, print the root element, and print the depth of the maximum complete tree rooted at that node.



The largest complete subtree of this binary tree has depth 2.

DETAILS

The starter code provides a way of demonstrating the various routines work. **Feel free to modify prototypes if something else is more useful.**

Each node includes a parent pointer. Make sure that parent pointers are correct in any tree you create.

You will notice in the starter code I have public helper functions, which make it possible to call a routine without knowing the root, but have recursive "worker" routines that depend on knowing the current node. That is a common practice.

In the comments to each function, provide a big-Oh expression for the complexity of the functions you write, assuming trees are roughly balanced (depth = log(n) for n nodes). Use recursion where appropriate, but if something isn't logically recursive, don't use recursion.

If you need to return MORE than one thing, you can do that by using references. So for example, if wanted to find the longest increasing sequence in an array and need the subscript at which it begins and the subscript at which it ends, you could use the following prototype:

```
void LongestSeq(int a[], int & begins, int & ends)
```

Hints

In BuildFromPreorder, I found it useful to pass the array pointer by reference. Then, to skip over what I had used (before calling recursively), I incremented the pointer (preorder=preorder+2).