CSC 212: Data Structures and Abstractions
Fall 2018
University of Rhode Island

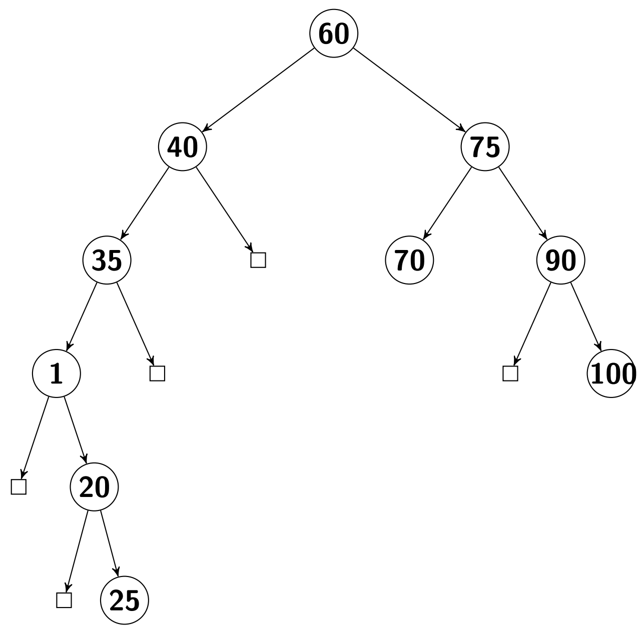
Weekly Problem Set #9

This is an optional practice assignment!

1 Binary Trees

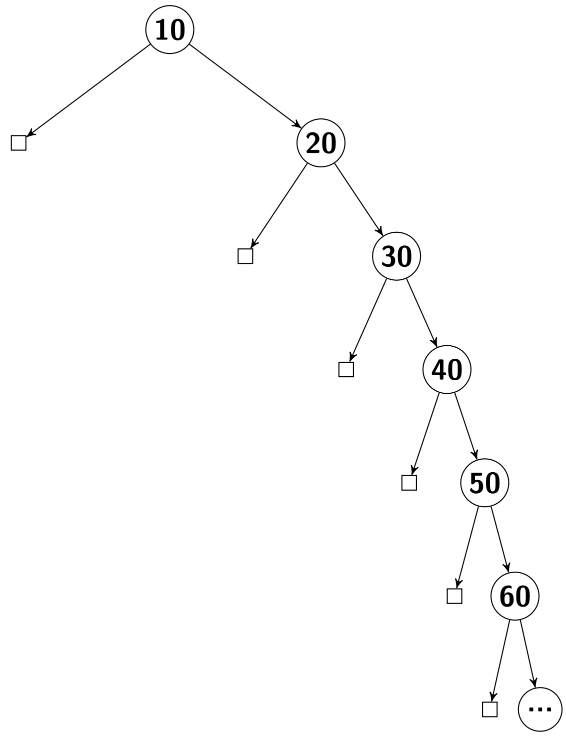
1. Draw a binary tree after the insertion of the following elements in order:

   `[60, 40, 35, 75, 90, 1, 20, 100, 25, 70]`



2. Draw a binary tree after the insertion of the following elements in order:

   `[10, 20, 30, 40, 50, 60, 70, 80, 90, 100]`

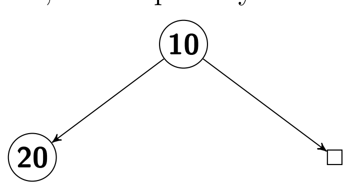


3. Explain what differs in the above two trees. Specifically, address how many leaf nodes there are, the depth of the right, and leftmost nodes, and the height of both the left and right subtrees from the root node.

   The tree with elements inserted in sorted order only has one leaf, whereas the previous tree had 3. The sorted tree is also extremely imbalanced, with all elements going towards the right.

4. If a Binary Tree is complete, does that necessarily mean it is also full? Justify your answer with drawings of trees.

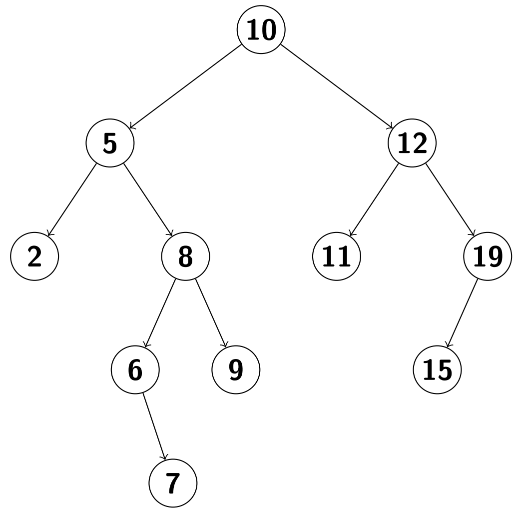
   No, the deepest layer of the tree could have a node that only possesses a single child.



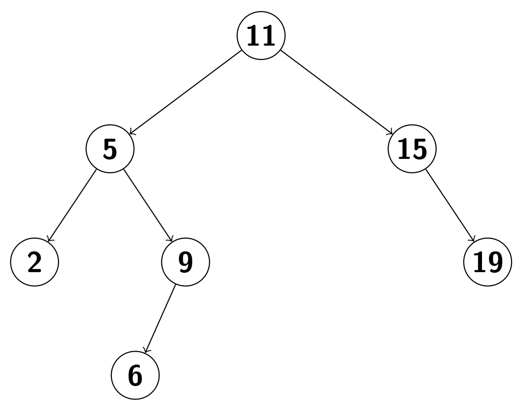
   Is complete, but not full.

5. Draw a binary search tree after the following operations steps:

   (a) Insert: [10, 5, 12, 8, 19, 6, 2, 11, 15, 9, 7]



(b) Remove: [7, 12, 8, 10]



6. Write a function to delete binary trees. Be sure to remove nodes in the proper order, so that none get orphaned.

```
// Initial call should pass root to clear whole tree.
void clear(BSTNode* node) {
    if (!node) return;

    clear(node->left);
    clear(node->right);
    delete node;
    return;
}
```

7. Assume nodes in a BST contain 4 data members: *data, depth, left, right.* Write a recursive function that, given a pointer to the root of a BST, will update every node's *depth* to it's own depth in the tree.

```
// Initial call should pass 0 for 'depth'.
void update_depth(BSTNode* node, int depth) {
    if (node) {
        node->depth = depth;
        update_depth(node->left, depth+1);
        update_depth(node->right, depth+1);
    }
}
```

8. Briefly explain the difference between in-order, post-order, and pre-order traversals.

   The three terms above signify different methods of traversing a binary tree. From the point of view of the node, in-order signifies visiting the left child, then itself, then the right child; post-order signifies visiting both children before itself; pre-order signifies visiting itself before any children. All three are necessary for different scenarios, as all provide different utility. For example, you cannot delete a node until all children are removed, yet you cannot know the depth of the children without visiting the parent first.

9. Let T be a full k-ary tree, where $k = 2$ (a.k.a. *binary tree*), with n nodes. Let h denote the height of T.

   (a) What is the minimum number of leaves for T? Justify your answer.

$$h + 1$$

   Example when $h = 0$: T, being a *full tree* can have a minimum of 1 leaf.

(b) What is the maximum number of leaves for T? Justify your answer.

$$2^h$$

If we consider the root having height 0, then our maximum number of leaves is when we have a perfect binary tree of height h, thus the maximum number of leaves we can have is:

$$1, 2, 4, 8, \ldots, 2^h$$

(c) What is the minimum number of internal nodes for T? Justify your answer.

$$h$$

Considering the smallest tree, with only one node, we can see that it is possible to have 0 internal nodes.

(d) What is the maximum number of internal nodes for T? Justify your answer.

$$2^h - 1$$

The maximum number of internal nodes can be expressed as the difference between the number of nodes in a tree, and the number of leaves (since nodes can either be leaves, or be internal). The maximum number of internal nodes occurs when a tree is perfect. The number of nodes in a perfect tree is expressed as $2^{h+1} - 1$, the number of leaves in a perfect tree is expressed as 2^h, therefore:

$$(2^{h+1} - 1) - 2^h = 2^h - 1$$

10. Give a $O(n)$ time algorithm for computing the `height` of the tree, where n is the number of nodes.

```
// Initial call should pass depth=-1, this special value
// is returned when the tree is empty.
int height(BSTNode* node, int h) {
    if (node) {
        return max(
            depth(node->left, depth+1),
            depth(node->right), depth+1)
        )
    } else {
        return depth;
    }
}
```

11. Show that the maximum number of nodes in a binary tree of height h is $2^{h+1} - 1$.

$$1 + 2 + 4 + 8 + \ldots + 2^h = 2^{h+1} - 1$$