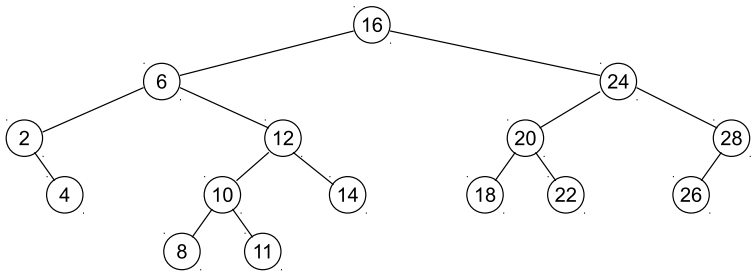
ECE-250 – Algorithms and Data Structures (Winter 2012)

Tutorial 5 (2012-02-16)

1 - Given the AVL tree in the following figure:



(a) Remove the smallest element twice, then remove the largest element twice, then remove the root node twice.

(b) After completing all removals from part (a), insert the following values in the given order: 17, 1, 15, 36, 41.

2 - Write a recursive C++ function that determines whether a given binary tree has a configuration consistent with a valid AVL tree. The function declaration or function prototype would be as follows:

```
template <typename Type>
bool is_avl (const Binary_node<Type> * tree);
```

3 - Discuss (and sketch the modified function) how could you make this function more efficient, by avoiding redundant calculations of the height of sub-trees. (Hint: how about the recursion going bottom-to-top, and given the height of the lower sub-trees, we compute the height of the sub-trees corresponding to the parent — maybe the function could return an `int` instead of just `bool`?)

4 - Given a binary search tree, which of the four traversals (breadth-first, pre-order, in-order, and post-order depth-first) will list the entries in such a way as to reconstruct the same tree contents if they are inserted into an initially empty binary search tree in the order in which they're visited with the given traversal? (for the yes cases, explain why, and for the no cases, show a counter-example to support your case)

How would the answer change if we consider an AVL tree instead of simply a binary search tree? That is, if we have an AVL tree and would like to insert the values into an initially empty AVL tree in such an order as to recreate the same AVL tree?