Worksheet 32: Tree Sort

In Preparation: Read chapter 10 on the Tree data type.

Both the Skip List and the AVL tree provide a very fast O(log n) execution time for all three of the fundamental Bag operations: addition, contains, and remove. They also maintain values in sorted order. If we add an iterator then we can loop over all the values, in order, in O(n) steps. This makes them a good general purpose container.

Here is an application you might not have immediately thought about: sorting. Recall some of the sorting algorithms we have seen. Insertion sort and selection sort are $O(n^2)$, although insertion sort can be linear if the input is already nearly sorted. Merge sort and quick sort are faster at O(n log n), although quick sort can be $O(n^2)$ in its worst case if the input distribution is particularly bad.

Consider the following sorting algorithm:

How to sort a array A
Step 1: copy each element from A into an AVL tree
Step 2: copy each element from the AVL Tree back into the array

Assuming the array has n elements, what is the algorithmic execution time for step 1? O(n log n)

What is the algorithmic execution time for step 2? O(n)

Recall that insertion sort and quick sort are examples of algorithms that can have very different execution times depending upon the distribution of input values. Is the execution time for this algorithm dependent in any way on the input distribution? No, the elements don't need to be sorted before beginning the algorithm, but an in-order traversal in step 2 will produce a sorted array

A sorted dynamic array bag also maintained elements in order. Why would this algorithm not work with that data structure? It would not work because every add or remove shifts the elements in the bag in such a way that the array would remain in order. In theory, it would work, but step 2 would be incredibly inefficient.
Also, because the array is already sorted and binary search is already O(log n), implementing tree sort with an ordered bag would render both ADTs cumbersomely inefficient.

What would be the resulting algorithmic execution time if you tried to do this?
Step 1: O(n log n)
Step 2: $O(n^2)$

Can you think of any disadvantages of this algorithm?
Yeah, both tree sort and ordered bag ADT are much more efficient on their own, and they both create an ordered array. However, combining the two would be as efficient as insertion sort. It would be more efficient to run the algorithm on a simple, unordered array or to maintain an ordered bag (and have an efficient binary search infused contains function of O(log n) and acceptable insertion/removal functional algorithms of O(n).

The algorithm shown above is known as tree sort (because it is traditionally performed with AVL trees). Complete the following implementation of this algorithm. Note that you have two ways to form a loop. You can use an indexing loop, or an iterator loop. Which is easier for step 1? Which is easier for step 2? Step 1 – indexing loop, step 2, iterator loop.

```
/*****************************************************************************************************
*  sort values in array data
*****************************************************************************************************/
void treeSort (TYPE *data, int n)  {

        //initialize a new AVL tree
        AVLtree tree;
        AVLtreeInit (&tree);

        //step 1, copy array to AVL tree
        for (int i = 0; i < data->size; i++)
                AVLaddNode(data[i]);

        //step 2, in-order traversal of AVL tree
        TYPE *array = data;
        dyArrayClear; //clear array to simplify adding elements in sorted order w/o overwriting
        inOrderAVLsort(&tree.root, array) ;
}


/*****************************************************************************************************
* recursively copy values to array during in-order traversal of AVL tree
*****************************************************************************************************/
void inOrderAVLsort (struct Node *node, TYPE *array)  {

        if (node != 0) {
                inOrderAVLsort (&node.right, array);
                dyArrayAdd (&node.value);
                inOrderAVLsort (&node.left, array);
        }
}
```