Group 17 Worksheet 30
Tanner England, Eddie Fox, Trevor Worthey, Lok Chan, Catherine Smith

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? What is the algorithmic execution time for step 2? 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?

1. O(nlogn)
2. O(n) with iterator as mentioned above
3. We don't believe input distribution would have an impact on step 2 of this algorithm but it could potentially lower the addition execution time of step 1 closer to O(n) from O(nlogn) – however, the chances of this sort of distribution are low

A sorted dynamic array bag also maintained elements in order. Why would this algorithm not work with that data structure? What would be the resulting algorithmic execution time if you tried to do this?

1. Why would you call another algorithm to sort an already sorted list? If this algorithm was implemented using a sorted dynamic array bag, its execution time would be close to O(n^2) – the addition complexity would be close to O(n) making step 1 almost O(n^2) – it would be terribly inefficient in comparison to the other ADL's mentioned above. The AVL tree would have to balance after almost every addition.

Can you think of any disadvantages of this algorithm?

1. We are starting with an array, then adding each element to an AVL tree, then copying each element from the AVL tree back into the original array. We are using twice as much memory as would other sort methods which sort in place such as quick sort.

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?

```
void treeSort (TYPE *data, int n) { /* sort values in array data */
   AVLtree tree;
   AVLtreeInit (&tree);
   for(int i = 0; i < n; i++)
          addAVLTree(&tree, data[i]); //add array values to tree
   BSTIterator *iter;
   BSTIteratorInit(&tree, iter); //initiate iterator
   Type *val;
   int i = 0;
   while(BSTIteratorHasNext(itr)) {
       val = BSTIteratorNext(itr);
       if(!val) //make sure something didn't go wrong
                   return;
       addAtDynArr(data, i, val->value); // add back to array
       i++;
    }
}
```