1. We noted in class that binary search trees whose items have the same set of keys can look completely different. For example, one can be long and thin, while the other can be short and fat. What affects their topology (i.e., their “shape”) is the sequence in which the items are inserted. Now, we want to know if this is still the case for AVL trees. After all, AVL trees can’t be long and thin anymore.
   a. First, insert the following sequence of keys into an initially empty AVL tree: 1, 2, 3, 4, 5, 6, 7.
   b. Now, address the problem – if you had started off with a different sequence, will the topology of the tree stay the same or different? Why?

2. Suppose each node $v$ in the binary search tree $T$ has to also maintain a field $v.size$, the number of items stored in the subtree of $v$.
   a. Describe the modifications that have to be made when an item is inserted into $T$ or removed from $T$.
   b. Suppose $T$ is also an AVL tree. Describe the additional changes have to be made to maintain $v.size$ properly at each node $v$.

3. Let $D$ be an ordered dictionary ADT. Let $\text{findAllElements}(k)$ be a method that returns all the elements stored in $D$ whose key is equal to $k$. Describe a procedure for implementing $\text{findAllElements}(k)$
   a. when $D$ is implemented with a sorted array in $O(\log n + s)$ time, and
   b. when $D$ is implemented with a binary search tree in $O(h + s)$ time
   where $n$ is the number of items stored in $D$, $s$ is the size of the output and $h$ is the height of the binary search tree.

4. A concatenate operation takes two sets, such that all keys in one set are smaller than all the keys in the other set, and merges them together. Suppose $T_1$ and $T_2$ are binary search trees where all the keys in $T_1$ are smaller than all the keys in $T_2$. Design an algorithm that concatenates $T_1$ and $T_2$ into a single binary search tree. The worst case running time should be $O(h)$, where $h$ is the maximum of $h_1$ and $h_2$, the heights of $T_1$ and $T_2$. 