1. Show that the time to compute the successor of an element in a binary search tree by the algorithm on page 249 of CLR requires $O(h)$ time in the worst case. Show that if you augment the tree element data structure with $x.successor$ and $x.predecessor$ links for each tree element, $x$, (called $x.su$ and $x.pr$ to make the code more readable) then
   a. You can find the predecessor and successor of a tree element in $O(1)$ time (DUH!).
   b. The links can be updated whenever there is an insert or delete in $O(1)$ extra time, even if the insert or delete itself takes $O(h)$ time. (hint: each element has a unique predecessor and successor).

2. We start with a binary search tree, $T$, and delete $x$ then $y$ from $T$, giving a tree, $T'$. Suppose instead, we delete first $y$ then $x$ from $T$. Do we get the same tree $T'$. Give a proof or a counterexample.

3. We wish to use a binary search tree data structure to implement a priority queue. Write a specialized binary search tree version of deleteMax for this purpose.

4. A family of trees has “bounded imbalance” if there is a single constant, $C$, so that $h \leq C \log_2(n)$ for each tree in the family. If there are many trees in the family, we have to use the $h$ and $n$ for each tree (but never the $h$ from one and the $n$ from another). Suppose you start with a complete binary search tree of height $k$ containing $2^k - 1$ numbers and then do $2^k - 2$ deleteMax operations. Does the family of trees produced have the bounded imbalance property? Either prove that it does or prove that it does not.