4. **(35 points) Balanced Binary Search Trees**

(a) (2 point) In your own words, list the properties of a **Balanced Binary Search Tree** (BBST). Use terminology discussed in class.

A Balanced BST has the same properties as a BST, its rights subtree should be larger than root and left smaller than Root

However in a Balanced BST there Balance Factor of each node or height of left and right should not be greater than 0 or 1 and there is a series of rotations that take place where each node is rotated left or right to keep tree balanced

(b) (3 points) What are the respective asymptotic worst-case run-times of each of the following operations of a BBST? Give a Θ bound if appropriate. Do not forget to include the complexity of the rebalancing operation where needed. Justify your answers. You do NOT need to do a line-by-line analysis of code.

i. Insert
ii. Delete
iii. Find-next
iv. Find-prev
v. Find-min
vi. Find-max

i.) In worst case an BBST insert will take place in $O(\log n)$ time as we move down the tree $\frac{n}{2}$ times, n = # of elements

ii) In worst-case delete takes $O(\log n)$ time as we have to move down the tree to find the desired node and we do it $\frac{n}{2}$ times, n = # of elements

iii) Find - next & iv) Find - prev

Since we traverse down the the tree to get desired node and go $\frac{n}{2}$, n = # of elements times our worst-case is also $\Theta(\log n)$ in best case we have on element but we still have to check right and left

ii) Find min & Find - max both would ran in $O(\log n)$ time as we only look at one side of the tree, which takes $\frac{n}{2}$ time, n = # of elements

5. **(15 points) Constructing Trees**

(a) (5 points) (*You must submit code for this question!*) Use your **recursive implementations** of your AVL Tree and BST from Parts 1 and 2 to construct trees using `getRandomArray(10,000)`. Both trees must be made from the same array. In other words, **do not call the method twice - store the output of the method from `getRandomArray(10,000)` once and use it to construct both trees.**

(b) (5 points) Did you run into any issues? Test your code on a smaller input (say, `getRandomArray(10)`, and see if you're still running into the same error. If it works on inputs of size 10 but not size 10,000, your code is probably fine and this is expected! Explain why you're running into issues (or might run into issues), using concepts we covered in class.

We might run into issues for and large input because each time we make a recursive call we allocate some space on our call stack and run through the insertion n * n times as even if we get some large number and worst case go to look at one side of the array we still make a call to the other side, this runs in $O(n^2)$ time and eventually we run out of memory, this causes the program to crash.

(c) (5 points) (*You must submit code for this question!*) Use your **iterative implementations** of your AVL Tree and BST from Parts 1 and 2 to construct trees from the input of your implementation of `getRandomArray(10,000)`. Both trees must be made from the same array. In other words, **do not call the method twice - store the output of the method from `getRandomArray(10,000)` once and use it to construct both trees.**