1. **(20 points) Binary Search Trees**

   (a) (2 points) In your own words, list the properties of a Binary Search Tree (BST).

Properties of BST:

- A BST (Binary Search tree) has a node whose left child must be less than its root and right node that has a value greaterthan it.

- The left subtree should also be less than the root and right subtree greater than it

- Both subtrees must be binary search trees themselves

(b) (2 points) What are the respective asymptotic worst-case run-times of each of the following operations of a BST? Give a Θ bound if appropriate. 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) Insert is inserting down the height of the tree unless we insert only the root so $O(\lg n)$

ii) Delete we have to recurse down the height of the the tree to find and delete node $\Theta(\lg n)$ for all times

iii) Find-next we have to go down our tree and find the next node otherwise start from the root and find ancestor in any case we do $\Theta(\lg n)$ operations

iv) Find-prev will be sames as find-next $\Theta(\lg n)$

v) Find-min will run in $O(n)$ time as we have to continuously go down the left n = height of the left sub tree

vi) Find-max will run in $O(n)$ time, n = height of right subtree

deleteRec - Frame Work

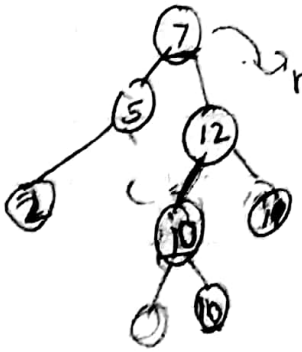
i) - We assume we take in a value to delete
- take in the root node as starting point
- remove the element from tree, we take in an integer

ii) One case: If we don't have any children for the node we want to delete the node

Second case: If we have one child we need to figure out which one it is and attach the parent of delete-node to its child

Also, think about duplicate values if they are in how they might affect our problem

3rd case: If there are two children we need to find the previous or next element and swap it with that, but we need to also see if the node we want to swap also has children



remove next is 10

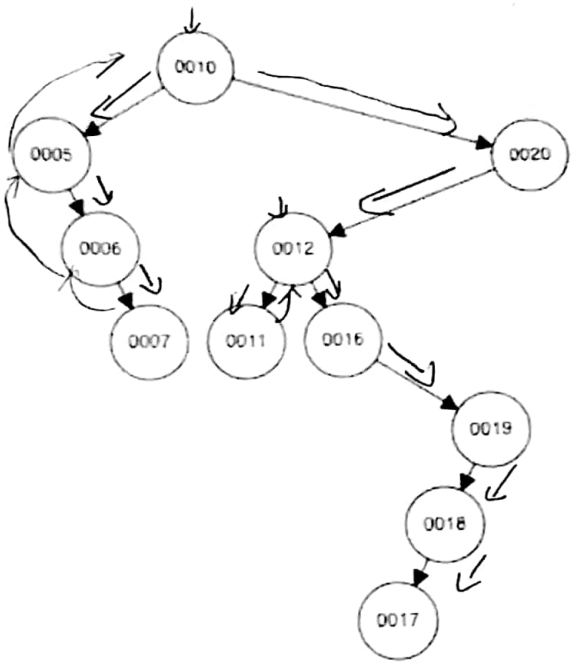
12 -> 10

v) Performance issue in my algorithm maybe where we swap our node to delete with a node that has children and how to point them to the parent.

) My current method has issue with deleting a node on the left subtree so I might need to trace it out to see where the code might not be handling the case, to optimize code. I might need to think about other edge cases than just the 3 and handle them for faster run time

2. **(10 points) Sort It!**

   (a) (1 point) In the following BST, what is the sorted order of elements, from lowest value to highest value? Write your answer as a comma-separated list.



0005, 0006, 0007, 0010, 0011, 0012, 0016, 0017, 0018, 0019, 0020

(b) (4 points) In your own words, describe an algorithm that uses the properties of a BST to take in a list of unsorted elements and output a list of sorted elements.

In BST we have our right subtree which is greater than root and left sub-tree less than the root While the subtrees are BST's themselves.

In our case we should first look at left side, sort it than go to right side and do the same

Since the subtrees themselves are BST's we implement same process

We basically do a inorder traversal where we sort each subtree and print it until we reach the end of the right side

(c) (5 points) (*You must submit code for this question!*) Implement the algorithm that you described above in `sort()`.