Q3

a)

Modify the BST to maintain a counter at every node. This counter would keep track of the # of children + 1 (for the node itself). On every insertion, set the counter of the node being inserted to 1, and increment the counter for every ancestral node.

When we perform a range search, we determine the boundary paths P1 and P2 (P1 being on the left side and P2 on right) as we would in a normal range search. Let's say we have a `total` counter which would indicate the number of points that lie in the range. When we go down P1, for each node on the boundary in our range, we add 1 for the node itself and the count stored on the direct right child for the inside nodes. Similarly while going down P2, for each node in our range, we add 1 and the count stored on the direct left child.

We can achieve a time complexity of `O(2log n) == O(log n)` as we need to iterate through the height of the tree twice for the two boundary paths.

b)

Say we have a balanced BST T of points (x, y) sorted by x-coordinates. Every node v in T has a balanced BST $T_y(v)$ associated with it consisting of and ordered by only the y-coordinates of the nodes in the subtree with root node v. Every T_y acts as the BST described in part (a) for the `counter` logic.

When we perform a range search, we determine the boundary paths P1 and P2 (P1 being on the left side and P2 on right) as we would in a normal range search. Let's say we have a `total` counter which would indicate the number of points that lie in the range. When we go down P1, for each node on the boundary in our range:

1. We add 1 for the node itself.
2. We also perform a search using our method from part (a) on T_y of the direct right child of the node v, with the range being the y-coordinates range, and add this to our total.

Similarly while going down P2, for each node in our range, we add 1 and the count using part (a) for T_y of the direct left child.

This works since we can determine the boundary nodes that fall in our range by doing simple comparisons. The x-coordinates of the inside nodes determined by this boundary fall in our range because our BST is sorted by x, however y-coordinates may or may not. Therefore we perform another search on the parent inside nodes to see how many of the y-values fall inside our range.

Similar to part (a), determining the boundary takes `O(log n)` time. For every boundary item that falls in our range, we perform another search that takes `O(log n)` time. Therefore the total cost is $O((\log n)^2)$.