We use S+1 BST, one for each specialty and one for all patients. Further we use a hash table to map patient id to information about his priority and tag. $$insert$$ and $$delete$$ operate on the tree of the patient specialty, the general tree, and the hash table.

$$find$$ operates on the hash table which takes O(1).

$$find\\_overall\\_highest\\_priority$$ operate on the general tree which takes O(logn). $$find\\_highest\\_priority$$ opretes on the corresponding specialty tree. $$update$$ calls delete and insert in a sequence to update information.

The data structure is similar to Q4, augmenting AVL tree at every node to store the number of nodes in its left tree and right tree. Apart from ordinary operations, $$insert$$ and $$delete$$ will update the number of nodes along the path to find the correct position or correct node. Maintain and rotate functions need to be overwritten to update $$num\\_l$$ and $$num\\_r$$. Such update takes constant time at each node, so the total runtime to build a tree is still O(nlogn). $$find$$ does not need to be modified.

num\_smaller(p):

cur = root, num = 0

while cur != NULL

if p < cur.value, cur = cur.left;

if p > cur.value, num+=1+cur.num\_l, cur = cur.right

if p = cur.value, num+=cur.num\_l, return num

num\_patients(p1,p2):

if find(p2) != NULL, return num\_smaller(p2) - num\_smaller(p1) + 1

else, return num\_smaller(p2) - num\_smaller(p1)