Complexity:

Search:

The complexity of search for a red-black binary tree is O(log(n)) in worst case. This is because in a red-black tree every leaf of a subtree has the same amount of black nodes up until the root. And red nodes can only be in-between two black nodes (nil is considered black). If the black nodes have the same depth for each leaf in a tree and there are red nodes only between the black nodes then the height of this tree can be maximum of log of black nodes+log of red nodes. Because the red nodes can only be inbetween black nodes we can say that we are guaranteed to have more black nodes than red we can say that this operation is upperbounded by 2*log(n) which is O(log(n)). The worst case is when the searched node does not exist in the tree or is in the bottom of the tree. The average case is same also O(log(n)) for red-black trees.

Insert:

Complexity of insertion is O(log(n)) for both worst case and best case for red-black trees. When searching for the place to insert the node inside the tree we spend log(n) time for the reasons specified in the previous paragraph. After the insertion when fixing tree to restore the red black property we only go up in the tree either one node or two nodes depending on the case which also takes time equal to the depth of the tree which is log(n) for worst case. In the best case our insertion does not disrupt the rb tree property but it still takes log-n time so we can say that the average time will also be O(log(n)).

BST vs red-black tree:

A regular binary search tree does not necessarily have to be balanced. If nodes are inserted uniformly into the tree the most probable case is for the tree to be balanced but this is the best case. On the other hand red black trees balance themselves after each insertion so that their depth never passes 2*log(n). A regular BST can have the depth n (a tree where all nodes have a single child) opposed to that which will cause it to have O(n) for deletion insertion and search in the worst case. Because of their self balancing properties the red-black trees are guaranteed to have O(log(n)) time complexity for previously mentioned operations.

Augmenting Data Structures:

If I wanted to be able to calculate i’th positions for all those roles I would keep amount of players from the specified position in the subtree for each position. To protect the child amounts for each position in their respective subtree I would have a function such as this:

```
//after a node is inserted
function update_rank_pg(node)
        if node is root return
        node.pg_amount+=1
        update_rank_pg(node.parent)
end
```

This function would update the upper nodes whenever a node is added into the tree achieving true values in all nodes. Keeping the amount of players in that position for that subtree in the node allows me to calculate the rank of the current node simply by adding one to the value in the left node if the current node does not have the required position. If the current node is not from the required position we should check left and right trees to deduce which way should we keep searching in. The pseudocode for this operation can be given such as:

```
function PG_i'th(node,rank)
        if node.role is PG and node.left.pg_amount+1 == rank then
                return node
        elseif node.left.pg_amount < rank then
                PG_i'th(node.left,rank)
        else
                PG_i'th(node.right,rank-node.left.pg_amount - (node.role is pg ? 1 : 0))
        end
end
```

In this function we recursively search for the node with I'th rank for pg just as if it was a regular order statistics tree. The only difference in this case is that our current node may actually not count for the ranking when we are searching the ranking in the right subtree (if the current node is not pg then when looking for the right subtree we only subtract amount of pg nodes in the subtree of left because current node does not count).

When doing the rotations we can do the same thing we do in order statistics trees which is to recalculate the values of the 2 rotated nodes from their children.