Report

NAME: BURAK ŞEN

ID: 150170063

1. **Complexity:**

Write down the asymptotic upper bound for the insertion and search operations of RedBlack Tree for worst case and average case with detailed explanations.

Insertion:

Because when we insert a node we go down from the root node and to a leaf node. We change the level of our node and compare our node with another node from the tree. With this operatşion we travel from the root and to NIL node, because between root and a NIL node there is **nlogn level** we can react the NIL node in **nlogn** steps, and this is our asymptotic upper bound for the Insertion operation.

Insertion asymptotic upper bound is O(nlogn).

Search Operation:

In the search operation our time complexity is the same with Insertion operation, because like in the Insertion operation we have to look nodes for each level, and we do know that our tree has nlogn levels of nodes, form this we can easily say that our time complexity for Search operation is O(nlogn).

Search asymptotic upper bound is O(nlogn).

2. **RBT vs BST**

In the BST when we insert a node from sorted list, we have the worst case scenario, because we want to reach every node as quick as possible. If our list is sorted all of our nodes goes to left of their parent adn we would have a single linked list instead of a BST structure (in visual). In this case if we use RBT we would break the linked list and we would have a proper tree and for each node we could have reach as quick as possible according to their level.

3. Augmenting Data Structures

I would probably search all the tree. Firstly I would create 5 individual counters for this. I would recursively visit each node and when I encounter with one of those cases, I would increase the counter of specific method. If the counter is 4 I would return that Node. To stop the recursive integration and returning the node I would create a flag that found if I found the node that I want this flag would be true and return the Node from the recursive method. We can use this pseudocode in our 5 methods.

a. Pseudo-Code

```
counter <- 0
found <- false
getpg(node, num):
    temp = null
    if not found:
        temp  <- getpg(node.left, num)
        if temp not null:
            found <- true
    if not found:
        temp <- getpg(node.right, num)
        if temp not null:
            found <- true

    if not found and node.position is pg:
        counter <- counter + 1

    if counter is 4:
        return node
    else:
        return null
```