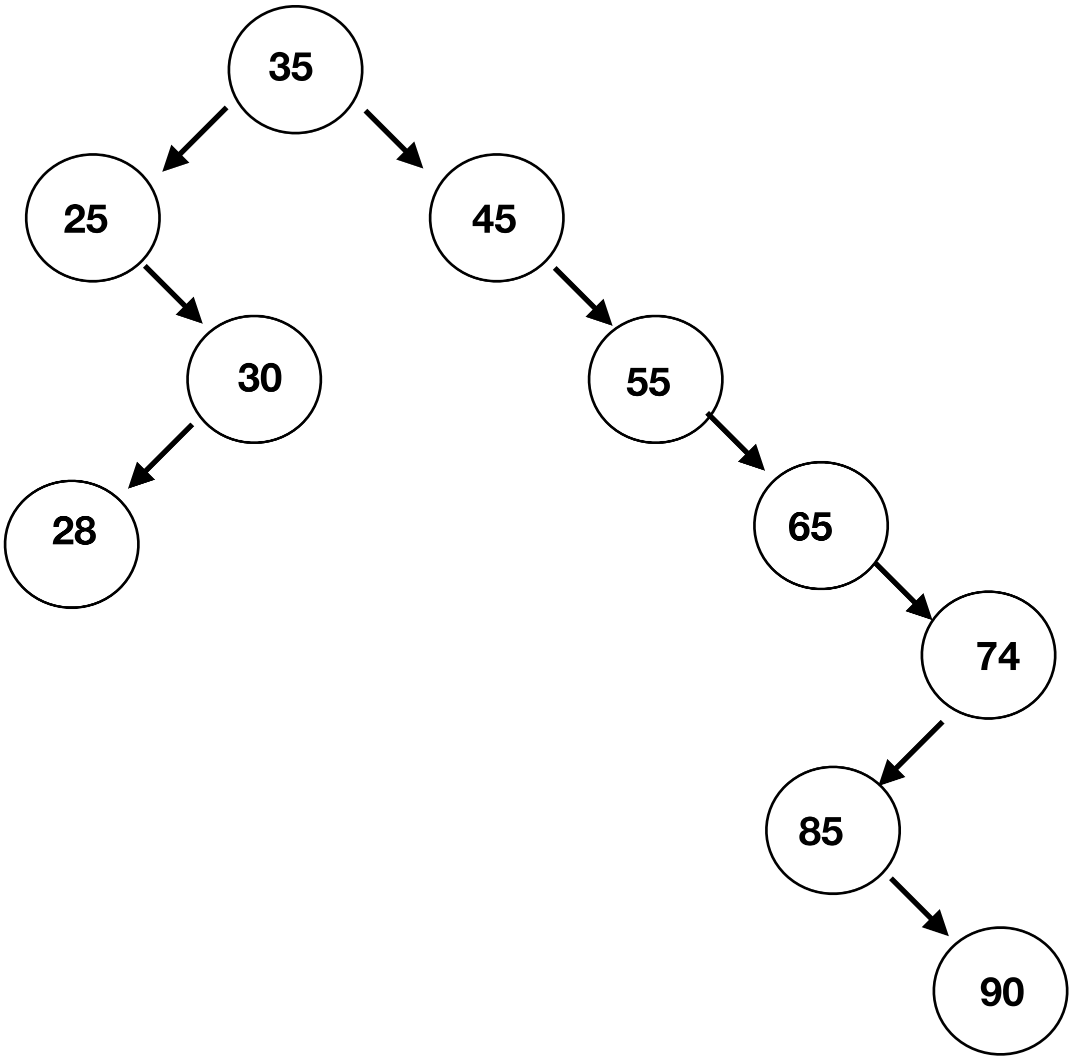
ANALYSIS OF ALGORITHM HW3 REPORT

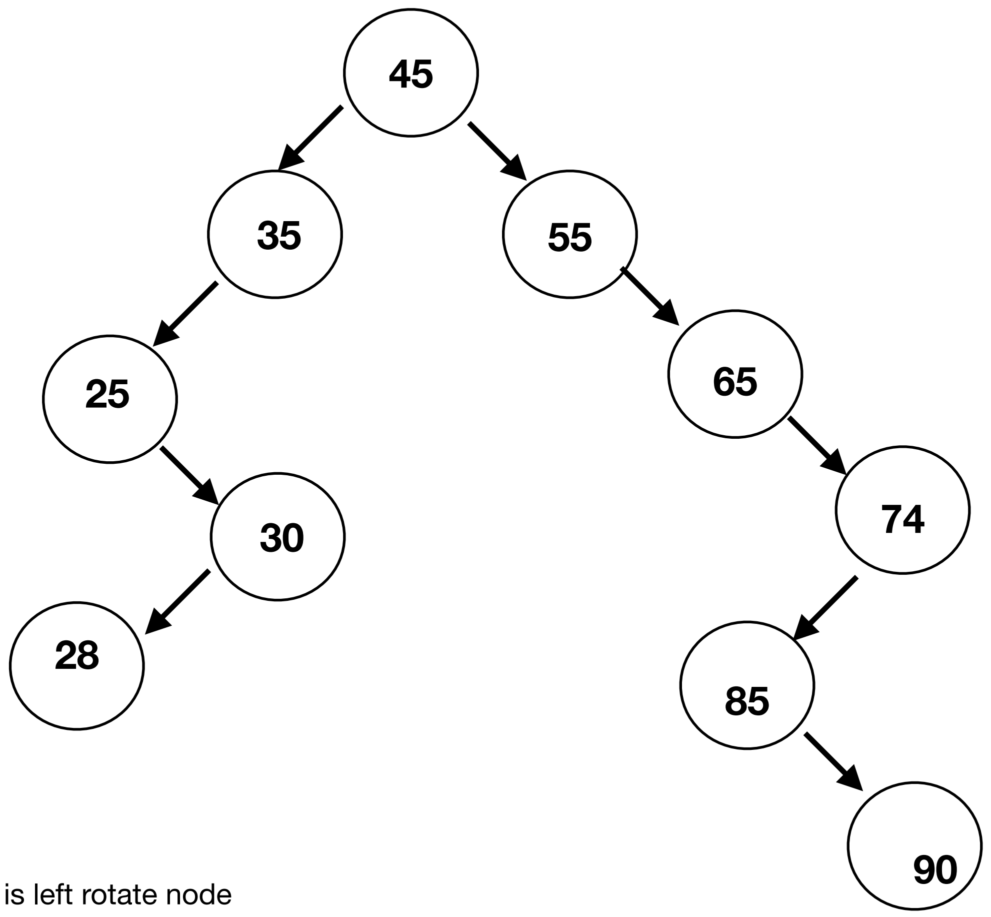
ELİF ARIKAN
150180010

RBT vs BST

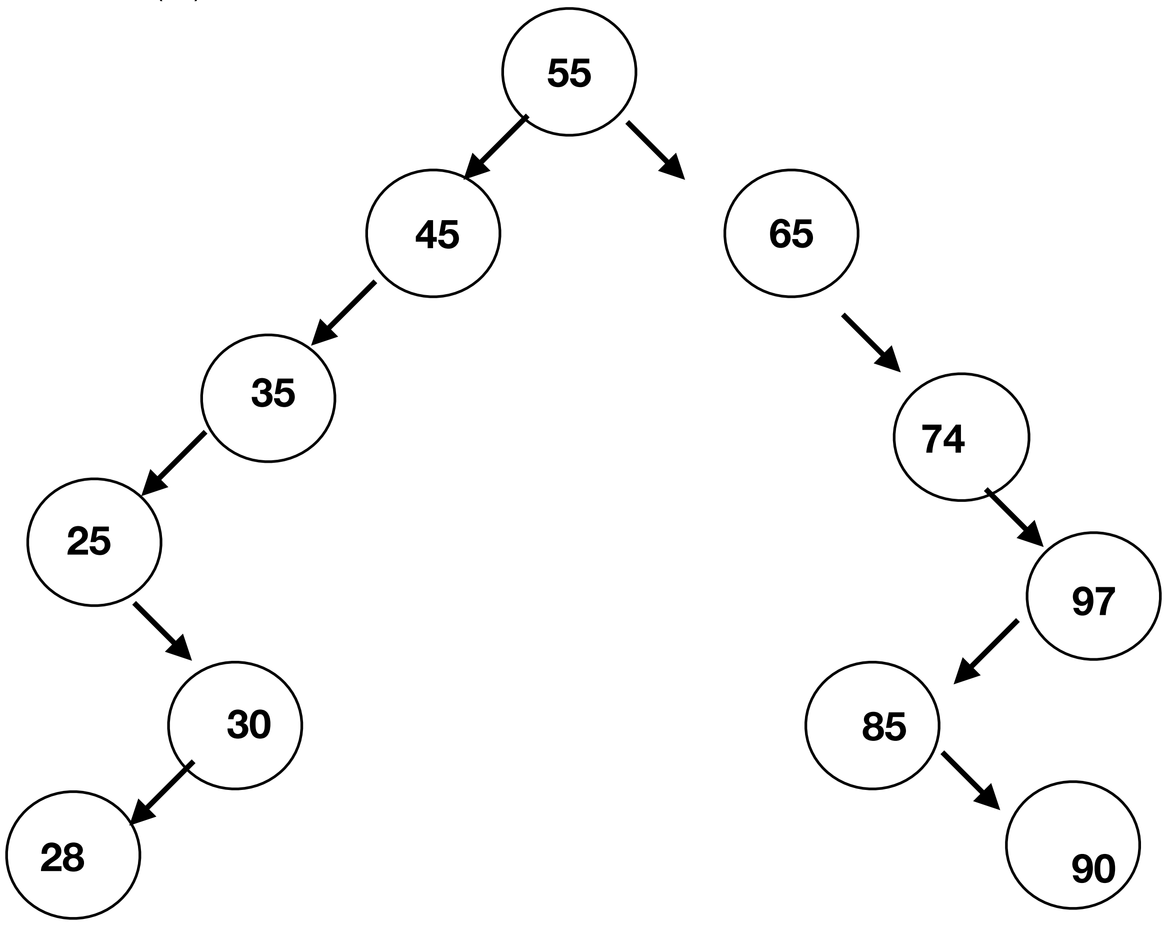
1. First of all, When I look at this binary search tree, I recognize that this bst is unbalanced tree. I will balance this tree step by step.



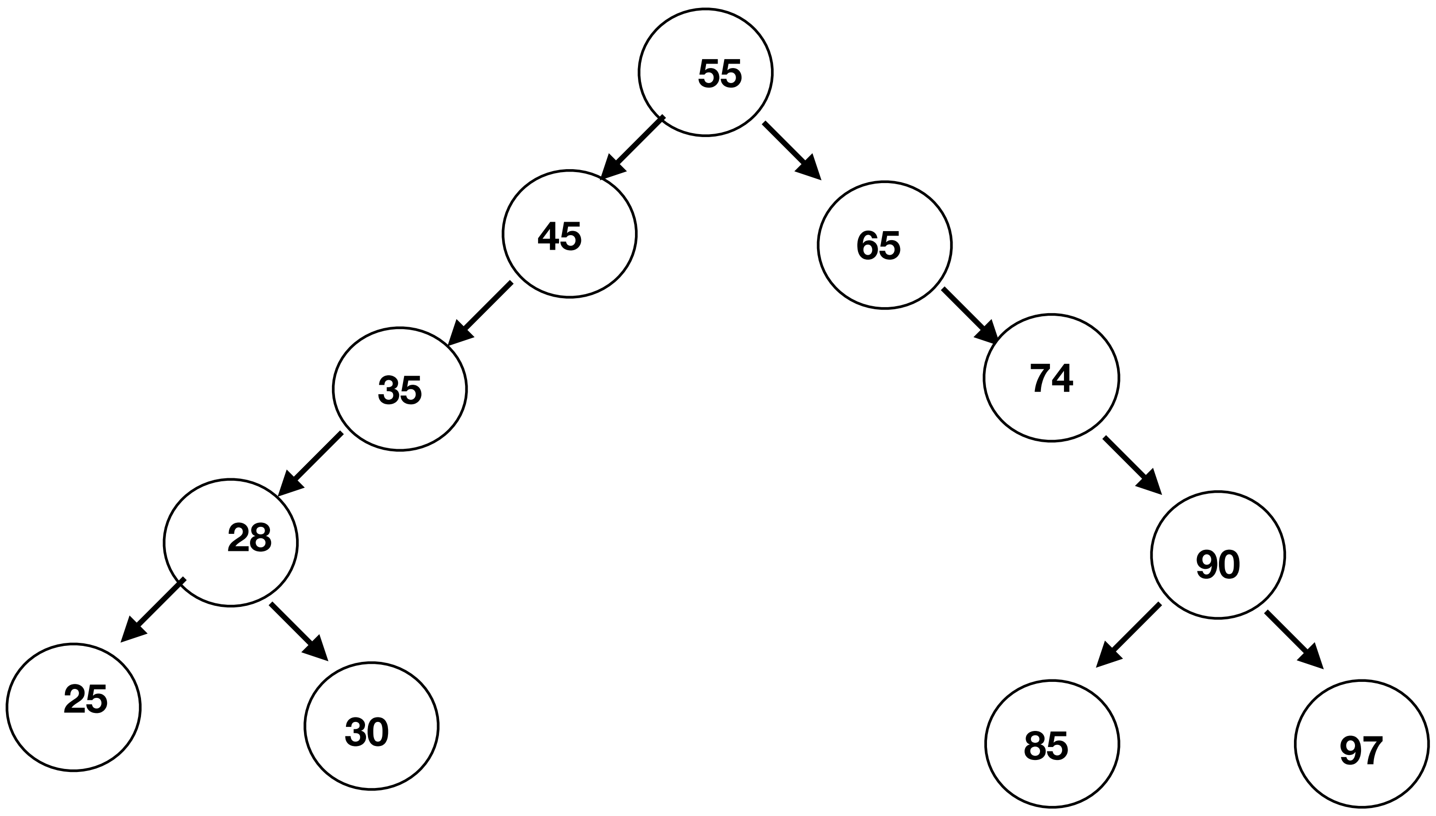
-first step left rotate node includes 45 .LEFT-ROTATE(45)



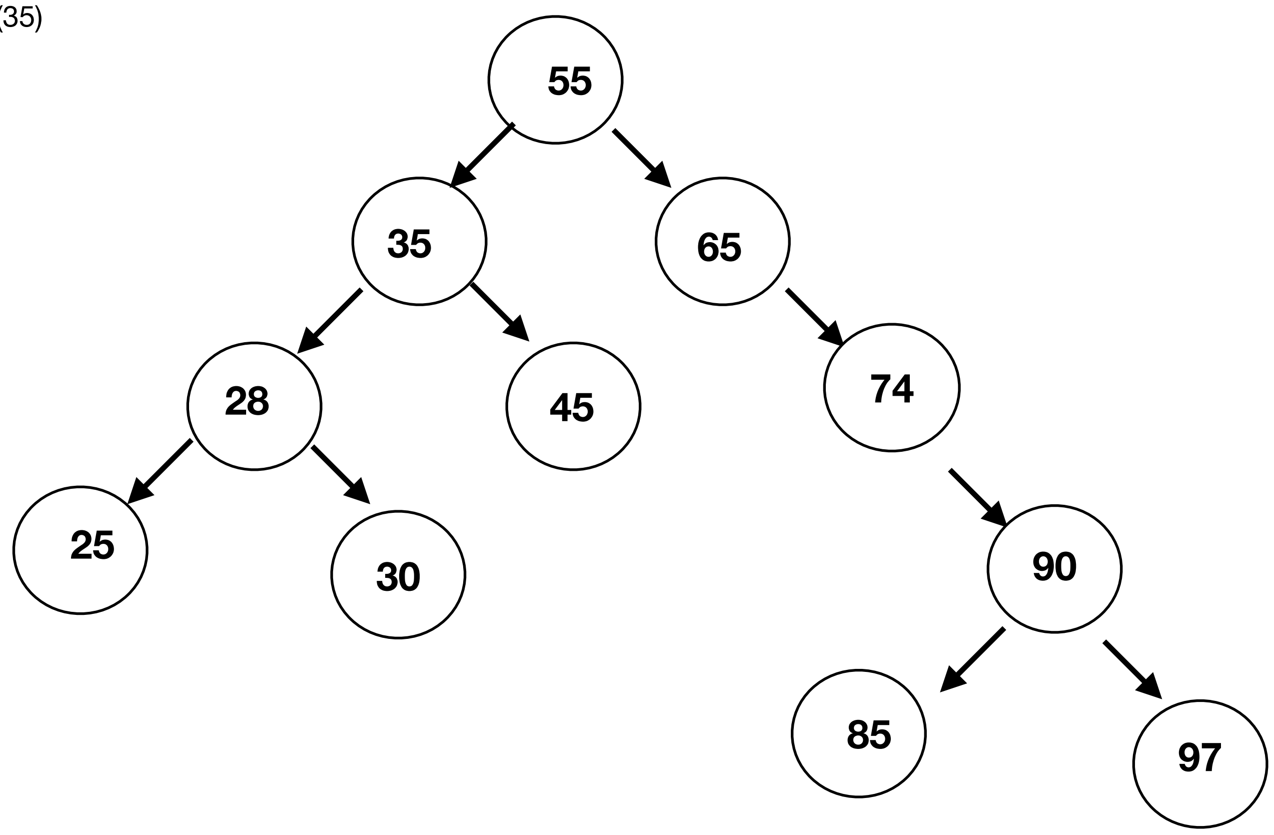
-Second step is left rotate node
includes 55.
LEFT-ROTATE(55)



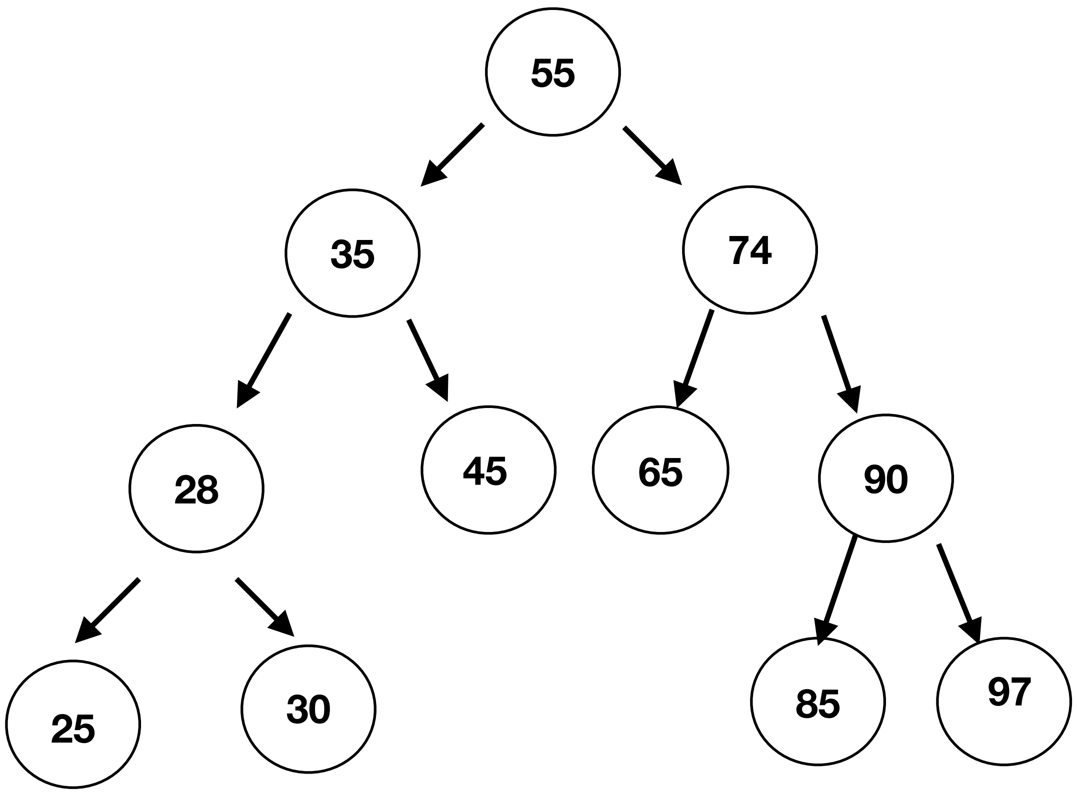
-third step Right-Left rotate node includes 30 ,RIGHT-ROTATE(30)+LEFT-ROTATE(28)
-and Left-Right rotate node includes 85, LEFT-ROTATE(85)+RIGHT-ROTATE(90)


$$\begin{tikzcd}
 & & & 55 \arrow[dl] \arrow[dr] & & & \\
 & & 45 \arrow[dl] & & 65 \arrow[dr] & & \\
 & 35 \arrow[dl] & & & & 74 \arrow[dr] & \\
28 \arrow[d] \arrow[dr] & & & & & & 90 \arrow[dl] \arrow[d] \\
25 & 30 & & & & 85 & 97
\end{tikzcd}$$

Fourth step is left-left rotate
node includes 35
RIGHT-ROTATE(35)


$$\begin{tikzcd}
 & & 55 \arrow[dl] \arrow[dr] & & \\
 & 35 \arrow[dl] \arrow[d] & 65 \arrow[dr] & & \\
28 \arrow[d] \arrow[dr] & 45 & & 74 \arrow[dr] & \\
25 & 30 & & & 90 \arrow[dl] \arrow[d] \\
 & & & 85 & 97
\end{tikzcd}$$

Fifth step is right right rotate node includes 74. LEFT-ROTATE(74).



2. Both BST and RBTres are basically the same in terms of BST's left node is smaller than the root, the right node is larger than the root. BST trees do not have self balance. RBTrees are balanced trees, and the complexity of the insertion, search, deletion time is O(logn) in either the best or the worst case. There is no guarantee that the insertion, search, deletion times will be O(logn) in BST, worst case O(n), best case is O(logn).

COMPLEXITY

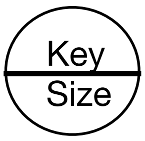
3. Operations like BST insertion search take O(h). The h we are talking about here is the height of the root node. A Red-Black Tree of height h has black-height >= h/2. If k is the minimal number of nodes on all root to NULL pathways in a generic Binary Tree, then n >= 2^k – 1. k = log2(n+1). In this situation k=height.

$$T(n)=O(h)=O(\log n)$$

In RBTree, all operations like insertion, search take O(logn) time for both of best-case and worst-case.

AUGMENTING DATA STRUCTURES

4.



size[x] = size[left[x]] + size[right[x]] + 1
size[NIL]=0
OS-SELECT(x, i) ith smallest element in the subtree rooted at x

```
OS-SELECT(x,i)
1  r ← size[left[x]] + 1                //r is the rank of node x within the subtree rooted at x
2  if i = r                             // If i = r, then node x is the ith smallest element, so we return x in line 3
3     then return x
4  elseif i < r                         //If i < r, then the ith smallest element is in x's left subtree,
5     then return OS-SELECT(left[x],i)  //so we recurse on left[x] in line 5
6  else return OS-SELECT(right[x],i - r) //If i > r, then the ith smallest element is in x's right subtree.
```