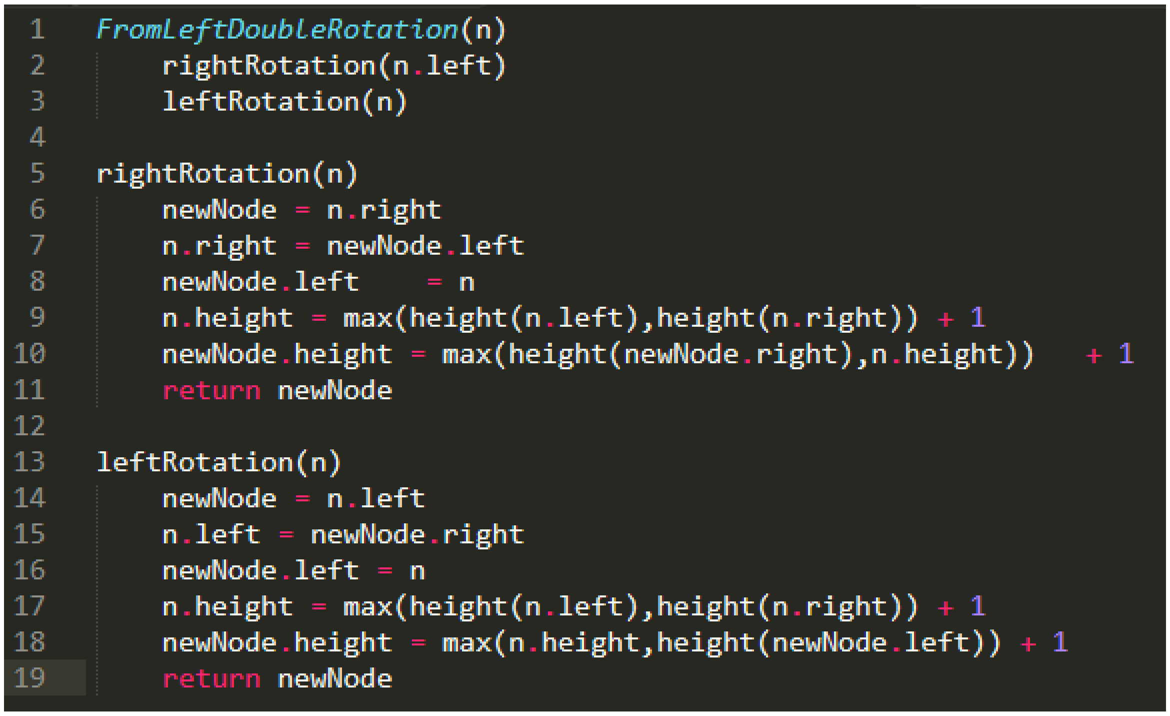
Aviad Hahami
302188347 Data Structures, Ex7

1)



```
FromLeftDoubleRotation(n)
    rightRotation(n.left)
    leftRotation(n)

rightRotation(n)
    newNode = n.right
    n.right = newNode.left
    newNode.left     = n
    n.height = max(height(n.left),height(n.right)) + 1
    newNode.height = max(height(newNode.right),n.height))    + 1
    return newNode

leftRotation(n)
    newNode = n.left
    n.left = newNode.right
    newNode.left = n
    n.height = max(height(n.left),height(n.right)) + 1
    newNode.height = max(n.height,height(newNode.left)) + 1
    return newNode
```

2) We will augment the tree in the following way:

Each node will contain the following data set:

- A field with the amount of nodes in his subtree +1 (himself)
- A field containing the sum of height in his subtree +1 (himself)

Insertion, deletion and AVL property maintenance still occurs in $O(\log(n))$ since we didn't change any property related data.

Heights calculation will be produced from dividing $\frac{SumOfHeight}{AmountOfNodes}$, hence equals $O(1)$.

To conclude, the total time consumption will be $O(\log(n))$

3) Let us assume that T_1's height is larger than T_2's

a) Algorithm:

i) Pick the smallest key in T_2 → $O(\log(n_2))$

ii) Since all the keys in T_1 smaller than T_2's keys (given), the most left key in T_2 is bigger than any key in T_1, hence we can connect T_1's root to the most left node in T_2.

--- Total cost → $O(\log(n_2))$

b) Algorithm:

i) Populate an array with T_1's keys using *inorder* algorithm, hence receiving a sorted array by the tree's property → $O(logn_1)$

ii) Invoke the algorithm from previous section over T_2 → $O(\log(n_2))$

iii) Merge the two arrays into sorted array → $O(n_1) + O(n_2)$

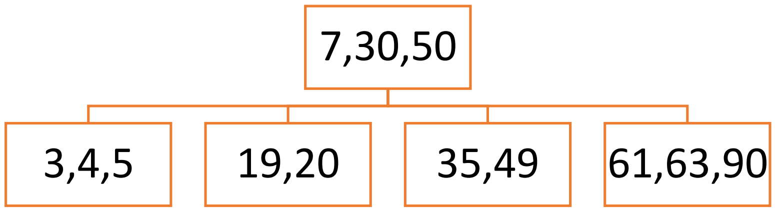
iv) Invoke the algorithm from Ex.6 (construct AVL from sorted array) → $O(n_1 + n_2)$

--- Total cost → $O(n_1 + n_2)$

4)

a) A B-tree's height is decreased whenever the root contains a singular key and his two childs have (t-1) keys. In our case we know that the tree's height decreased due to k's removal, implying that k was a singular key in the root. Implying that 2k is a key found in k's child nodes. Hence removing '2k' instead of 'k' will not decrease the tree's height.

b) For M=4:



Let's choose $K = 6$.
Inserting 6 will result in a tree modification as "6" will move up to the root.
On the exact same way, he will split the root too, resulting in height change.
Inserting 12 will cause no tree height modification as there is enough room in his designated node.

5)

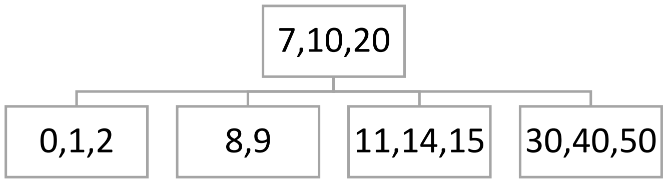
a) Amount of keys can be calculated via the following formula:

$$\frac{m}{2} - 1 \leq Keys \leq m - 1$$

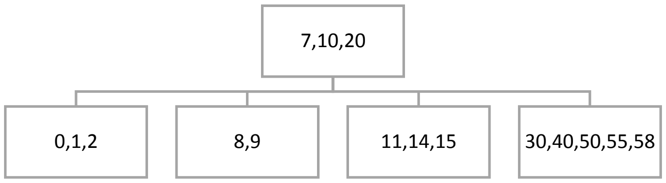
Hence minimum amount of keys is $\frac{6}{2} - 1 = 2$
Maximal amount of keys $6 - 1 = 5$

b)

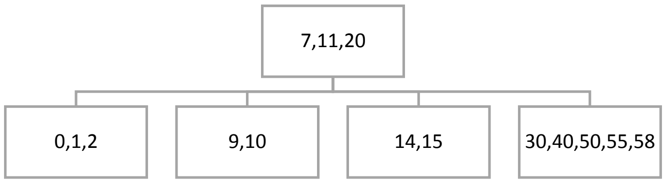
i) Inserting "0" will cause overflow in the left wing, hence a split will occur, raising 7 to the root.



ii) Inserting 58 & 55 will not cause any change since we can hold up to 5 keys.



iii) Removing 9 will cause underflow, hence demanding a unification.



c) In order to increase the tree's height we need to make his child nodes overflow, hence creating overflow propagation. We will use the following steps in order to create such propagation:
   i) Add {0} to the most left node, this will cause {1} to pop up to the root, adding one more child to the root. Add {3, 4, 5} respectively to fill up the node.
   ii) Add {16,17,18,19} to the node containing {11,14,15}, creating overflow, hence popping one number to the root.
   iii) Add the following to right most node$\{x, y, z \mid x, y, z > 20\}$, creating overflow in the rightmost node, hence popping up number to the root, hence propagating the overflow to the root, forcing new root & height change.

→ Minimum amount of keys to insert: 10

d) It is clear that in order to achieve a fully populated tree we need to fill up all existing node (5 keys each) and to add 3 more fully populated nodes (since M=6).
   Quick calculation will result in $3 + 2 + 2 + 3 * 5 = 22$ keys allowed to insert without changing the tree's height.

6)

a) We can sum this amount using the amount of nodes in each level times $m - 1$.

$$(m-1)*\left(1+m+m^2+\ldots+m^h\right)=\frac{\left(m^{h+1}-1\right)*(m-1)}{(m-1)}=\left(m^{h+1}-1\right)$$

Therefor the needed amount is $10^{13}-1$

b) Maximal height will be in the form of *Chain Graph*, and the height for such graph is $(n-1)$ therefor the max height for the previous number is $10^{13}-1-1=\ 10^{13}-2$.
Minimal height is in the case of fully balanced BST, and its' height is known to be $\log_2 n$, resulting in $\lfloor\log_2 10^{13}-1\rfloor$.