Date: 20/04/2019

Title: Heaps and AVL Trees

Author: Zeynep Cankara

ID: 21703381

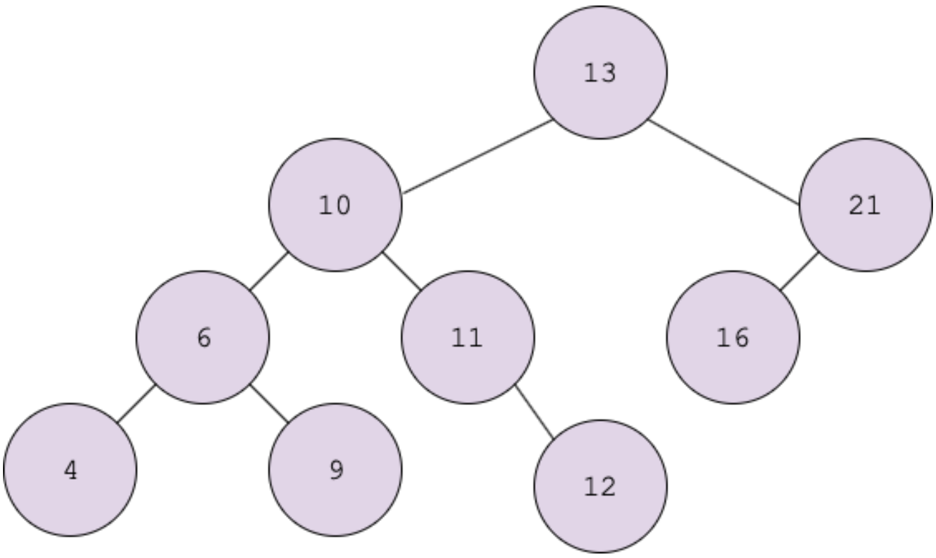
Section: 2

Assignment: 3

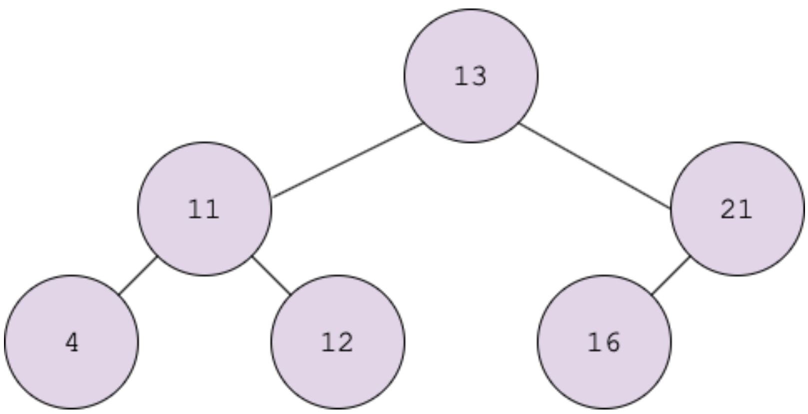
Description: My Solutions for Part1 a), b) and c).

Part 1: (a)

- After inserting 13,9,10,21,11,16,4,12,6 into an initially empty AVL tree in order fashion.

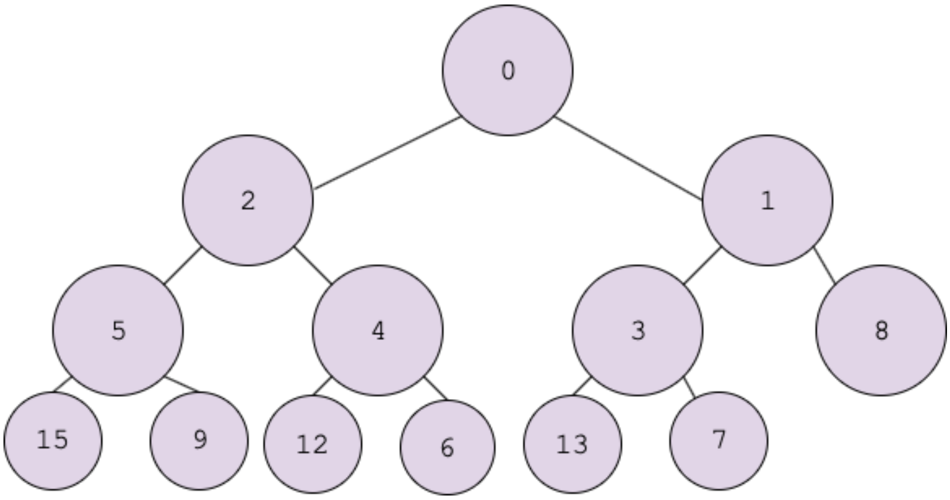


- After deleting 10, 9, 6 in given order from the AVL tree.



Part 1: (b)

- After inserting 15, 13, 9, 5, 12, 8, 7, 4, 0, 6, 2, 1, 3 into an empty min heap.



Part 1: (c)

JOIN-AVL- TREES(T1, T2) :

```
    // Get heights of T1 and T2 takes O(h1 + h2) time

    // Heights of AVL trees are logarithmic with respect to the
    // number of elements they have

    h1 = getHeight(T1);

    h2 = getHeight(T2);

    // assume that h1 >= h2 for simplification

    // delete node with the smaller item from T2

    // save the item in the smallest node

    delSmallest(T2, smallestT2); // takes O(h2)

    // now height of h2 becomes h

    // Find a node from T1 whose height h or (h+1)

    // takes O(h1) time

    node = getRoot(T1);
```

```
    curH = h_1;
    While curH > (h+1):
        balance = getBalance(node);
        if (balance == -1):
            curH = curH - 2;
        else
            curH = curH - 1;
        node = getRightChild(node); // go to the right child
    // get parent of node
    nodeParent = getParent(node);
/**
* Construct a new AVL-tree where 'smallestT2' becomes the
*  *root. T2' will become the right subtree of 'smallestT2' and
*  'node' will rooted at left subtree of 'smallestT2'. This
* construction will protect the AVL property. Now set
* 'nodeParent's right subtree to the root ('smallestT2').
* The new construction is still a binary search tree
* /
// Takes constant time
setRoot(smallestT2);
smallestT2.setLeftSubtree(node);
nodeParent.setRightSubtree(smallestT2);
// Starting from nodeParent check the balance factor do necessary
// rotations to fix AVL-property
// Just like insertion to a AVL-tree checking balance factor from
a node up to a root takes logarithmic time.
checkBalance(nodeParent);
```

Notes on justification algorithm takes $O(log(n))$ time:

Comments already explains how algorithm works, here is a summary on computational complexity of the algorithm.

STEPS:

- Computing height of an AVL trees T1, T2 $O(h_1 + h_2)$.
- Deleting a node from T2 $O(h_2)$
- Finding the node with height h or (h+1) $O(h_2)$
- Constructing a new AVL tree $O(1)$
- Fixing the BST to have AVL property $O(h_1)$