# Handson 1 - Anna Francesca Montagnoli

**Exercise 1**

*Write a method to check if the binary tree is a Binary Search Tree.*

I implemented the requested method using in-order (LNR) traversal and keeping track of the previously visited node. If the nodes visited are ordered in a non-decreasing key sequence then the tree is a BST.

1. If the node is null then the function returns true;
2. Calls recursively the method on the left subtree;
3. Checks if the previous node is greater than the current node and if it is so it returns false;
4. Sets the previous node to the current node;
5. Calls recursively the function on the right subtree.

Time complexity: O(n) with n = number of nodes.

**Exercise 2**

*Write a method to check if the binary tree is balanced.*
*A tree is considered balanced if, for each of its nodes, the heights of its left and right subtrees differ by at most one.*

The *rec_is_balanced* method visits recursively the tree and while doing so, it computes the height of every subtree to check if the difference between every left and right subtree is less than or equal to 1. The method returns -1 if the tree is not balanced and the height of the tree otherwise. The *is_balanced* method calls *rec_is_balanced* on the root of the tree and returns a boolean that indicates if the tree is balanced or not.

Time complexity: O(n).

**Exercise 3**

*Write a method to check if the binary tree is a max-heap.*
*A max-heap is a complete binary tree in which every node satisfies the max-heap property. A node satisfies the max-heap property if its key is greater than or equal to the keys of its children.*

The method I implemented works like this:

1. Check if the given tree is a complete binary tree by calling the function *is_complete*:

    - Count the total number of the tree's nodes using the function *count_nodes*;
    - Start the recursion on the binary tree, passing as the index argument the value 0;
    - If the node is null then the function returns true;
    - If the current index is greater than or equal to the number of the nodes return false;
    - Call the function for the left subtree with index 2*index+1 and right subtree with index 2*index+2

2. Traverse the tree to check for every node if the value of its children are less than or equal to its value.

Time complexity: O(n).