Handson 1 Massimiliano Baglioni

Exercise #1

The implemented solution makes use of the *in-order traversal* technique. If *in-order traversal* is executed on a BST, the resulting values will be returned in ascending order thus, to check if the given tree is a BST, it's enough to check that this condition is satisfied.

In particular two functions are implemented, ***"is_bst"*** where a variable used to keep track of the previous returned value is defined and where the other function "***is_bst_inorder"*** is called.

The second function just traverses the tree, in the in order fashion, and returns ***false*** if the condition of having the returning values in ascending order is not satisfied. The time complexity is $\Theta(N)$, used to traverse the tree. No additional data structure is used.

Exercise #2

The implemented solution uses two functions: ***"is_balanced"***, ***"is_balanced_rec"***. The first function is used just to return a boolean rather than a number which is clearer in my opinion.

Two variables keep track of the heights of the two subtrees. The returned value is the height of the tree (if the tree is balanced), otherwise ***"0"*** is returned. ***"1"*** is returned when a None value* is reached and then the evaluation is executed.

The evaluation consists in having a difference of at most ***"1"*** between the heights of the two subtrees. If this condition is not passed the returning value is ***"0"*** that will propagate until the end of the recursion. If the condition is passed then the highest height between the two subtrees is returned.

The time complexity is $\Theta(N)$, used to traverse the tree. No additional data structure is used.

*(this value is returned then when a leaf is reached or a when a None root is passed to the function)

Exercise #3

The implemented solution checks if the tree is complete using a separate function named ***"is_complete"***. Another function is used to check the max heap property. The return value is given by an AND operation between the return values of the two previous functions in the third function named ***"is_heap"***.

The ***"is_complete"*** function calculates the indexes that nodes have in the tree. Starting from an index of ***"0"*** for the root, the left child will have an index of $2 * parent\_id + 1$ and the right child will have $2 * parent\_id + 2$, if the tree is not complete at least one of this indexes will exceed the number of nodes in the tree.

The max heap check is pretty trivial, the parent node must have a bigger value than its children.

The time complexity is $\Theta(2N)$ since we traverse the tree two times, one to check if the tree is complete and another to check if the max heap property is satisfied.