CORK INSTITUTE OF TECHNOLOGY
INSTITIÚID TEICNEOLAÍOCHTA CHORCAÍ

Computing@CIT

ALGORITHMS AND DATA STRUCTURES

ASSIGNMENT 2: Operations on Binary Search Trees.
PART II: Recursion, Binary Search Trees.

1. Background.

The Abstract Data Type (ADT) myBinarySearchTrees<T1, T2> allows to store nodes with the format (T1 key, T2 value). Whereas T2 can be any datatype (e.g., an Integer, a String, a myPlayer, etc.), the datatype T1 is constrained to have a total order relationship ≤ (i.e., two elements T1 elem_i and T1 elem_j can be compared and sorted).

The ADT myBinarySearchTrees<T1, T2> supports the set of operations we have seen in the lectures, and it is specified in the interface myBinarySearchTree.java.

```
1.  //public myBinarySearchTree<T1, T2> create_from_binary_search_node(
                                        myBinarySearchNode<T1, T2> n);

2.  public boolean my_is_empty();
3.  public myBinarySearchNode<T1, T2> my_root();
4.  public myBinarySearchTree<T1, T2> my_left_tree() throws myException;
5.  public myBinarySearchTree<T1, T2> my_right_tree() throws myException;

6.  public myBinarySearchNode<T1,T2> my_find(T1 key);
7.  public myBinarySearchTree<T1, T2> my_insert(T1 key, T2 info);
8.  public myBinarySearchTree<T1, T2> my_remove(T1 key);

9.  public int my_length();
10. public int my_node_count();
11. public int my_leaf_count();
12. public myList<T2> my_inorder();
13. public myList<T2> my_preorder();
14. public myList<T2> my_postorder();
15. public myBinarySearchNode<T1, T2> my_maximum() throws myException;
16. public myBinarySearchNode<T1, T2> my_minimum() throws myException;
```

All the operations are implemented in the class myBinarySearchTreeImpl.java.

2. Goal of the Assignment.

In this assignment the ADT myBinarySearchTrees<T1, T2> has been extended with 3 new operations:

17. public int my_count_at_level(int level);
    This operation receives as an input the level of the tree we are looking for, and returns the amount of nodes placed on that level.

18. public boolean my_is_balanced();
    A binary tree is balanced when the length of its two subtrees (left subtree and right subtree) do not differ in more than 1 unit, and the proper left subtree and right subtree are balanced trees as well).
    This operation returns if the tree is balanced or not.

19. public int my_count_smaller_nodes(T1 key);
    This operation receives as an input a key, and returns the amount of nodes in the tree with smaller key values.

20. public int my_find_node_at_level(T1 key);
    This operation receives as an input a key, and returns the level were the node is located.

Exercise: Implement the 4 methods in the class myBinarySearchTreeImpl.java using recursion. *Note: You can reuse any of the other 16 defined operations if you want.*

3. Marking Scheme and Submission Date.

- **Total marks:** 10 (2.5 for each operation).
- **Submission instructions:** Submit to Blackboard the file myBinarySearchTreeImpl.java!
- **Submission deadline:** Sunday 7th May, 11:59pm.