CSC263 – Problem Set 2

Remember to write your **full name** and **student number** prominently on your submission. To avoid suspicions of plagiarism: at the beginning of your submission, **clearly state any resources (people, print, electronic) outside of your group, the course notes, and the course staff, that you consulted**.

Remember that you are required to submit your problem sets as both LaTeX `.tex` source files and `.pdf` files. There is a 10% penalty on the assignment for failing to submit both the `.tex` and `.pdf`.

Due Feb 11, 2019, 22:00; required files: ps2.pdf, ps2.tex, num_orders.py, num_trees.py

Answer each question completely, always justifying your claims and reasoning. Your solution will be graded not only on correctness, but also on clarity. Answers that are technically correct that are hard to understand will not receive full marks. Mark values for each question are contained in the [square brackets].

You may work in groups of up to THREE to complete these questions.

1. **[12]** Let $a_1, a_2, \ldots, a_n$ be a sequence of real numbers, for some $n \geq 1$. A `SUM-BOX` is an ADT that stores the sequence and supports the following operations (S is a given `SUM-BOX`):

   - `PARTIAL-SUM`(S, m): return $\sum_{i=1}^{m} a_i$, the partial sum from a_1 to a_m $(1 \leq m \leq n)$.
   - `CHANGE`(S, i, y): change the value of a_i to a real number y.

   Design a data structure that implements `SUM-BOX`, using an **augmented AVL tree**. The worst-case runtime of both `PARTIAL-SUM` and `CHANGE` must be in $\mathcal{O}(\log n)$. Describe your design by answering the following questions.

   (a) What is the key of each node in the AVL tree? What other attributes are stored in each node?

   (b) Write the pseudo-code of your `PARTIAL-SUM` operation, and explain why your code works correctly and why its worst-case running time is $\mathcal{O}(\log n)$. Let *S.root* denote the root node of the AVL tree.

   (c) Describe in clear and concise English how your `CHANGE` operation works, and explain why it runs in $\mathcal{O}(\log n)$ time while maintaining the attributes stored in the nodes of the AVL tree.

Programming Question

The best way to learn a data structure or an algorithm is to code it up. In each problem set, we will have a programming exercise for which you will be asked to write some code and submit it. You may also be asked to include a write-up about your code in the PDF/TeXfile that you submit. Make sure to **maintain your academic integrity** carefully, and protect your own work. The code you submit will be checked for plagiarism. It is much better to take the hit on a lower mark than risking much worse consequences by committing an academic offence.

2. **[12]** The function `num_orders` takes a list `lst` giving the insertion order of elements into an initially empty BST. For example, `[2, 1, 3]` means to insert 2, then insert 1, then insert 3. The function returns the total number of insertion orders (including `lst`) that produce the same BST that `lst` produces.

   Here is a sample call of `num_orders`:

   ```
   >>> num_orders([2, 1, 3])
   2
   ```

   The return value is 2 because there are 2 insertion orders, `[2, 1, 3]` and `[2, 3, 1]`, that produce the same BST as produced by `[2, 1, 3]`.

   Note that `lst` can contain duplicates. Let's agree that equal elements go into the left subtree (not the right subtree). For example, the root of the tree for the insertion sequence `[4, 4]` has 4 as its left node and an empty right subtree.

   Implement `num_orders`.

   Requirements:

- Your code must be written in Python 3, and the filename must be `num_orders.py`.
- We will grade only the `num_orders` function; please do not change its signature in the starter code. include as many helper functions as you wish.

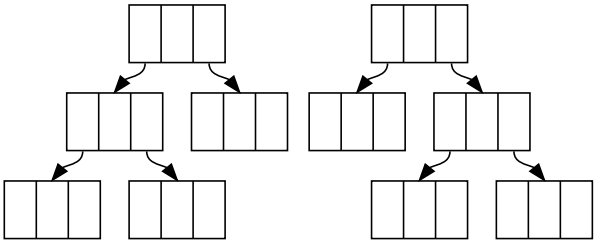
Write-up: in your `ps2.pdf/ps2.tex` files, include an explanation of how your code works. Please include a formal proof of correctness.

3. [**12**] The function `num_trees` takes the total number of `nodes` and the number of `leaves`, and returns the number of **AVL-balanced** tree shapes with that many nodes and leaves.

   Here is a sample call of `num_trees`:

   ```
   >>> num_trees(5, 3)
   2
   ```

   This means that there are exactly two AVL-balanced trees that have five nodes where three of those nodes are leaves. Here are those two trees:

   Implement `num_trees`.

   Note: we're not asking you to implement any optimizations. As such, this thing really slows down when the number of nodes increases. We hope that your code can solve cases with 8 nodes or fewer in under a minute. It should of course be correct for larger numbers of nodes too, but it's OK if the time taken in these cases is prohibitive. (We're happy to talk to you about several possible optimizations if you're interested!)

   Requirements:

   - Your code must be written in Python 3, and the filename must be `num_trees.py`.
   - We will grade only the `num_trees` function; please do not change its signature in the starter code. include as many helper functions as you wish.

   Write-up: in your `ps2.pdf/ps2.tex` files, include an explanation of how your code works. Please include a formal proof of correctness.