Worth: 8% **Due:** By 5:59pm on Tuesday 13 October

Remember to write the *full name* and *student number* of *every group member* prominently on your submission.

---

*Please read and understand the policy on Collaboration given on the Course Information Sheet. Then, to protect yourself, list on the front of your submission **every** source of information you used to complete this homework (other than your own lecture and tutorial notes). For example, indicate clearly the **name** of every student from another group with whom you had discussions, the **title and sections** of every textbook you consulted (including the course textbook), the **source** of every web document you used (including documents from the course webpage), etc.*

*For each question, please write up detailed answers carefully. Make sure that you use notation and terminology correctly, and that you explain and justify what you are doing. Marks **will** be deducted for incorrect or ambiguous use of notation and terminology, and for making incorrect, unjustified, ambiguous, or vague claims in your solutions.*

---

1. Consider the following abstract data type that we will call a "PIXELSET."

   Objects: A set S of "coloured pixels" that are represented by triples (x, y, c), where x and y are positive integers denoting a position of a pixel on the screen, and $c \in \{R, B, G\}$ denotes a colour. Note that each position can be assigned up to three colours, e.g., $(5, 3, R)$, $(5, 3, B)$ and $(5, 3, G)$ can co-exist in a PIXELSET, but there cannot be more than one "$(5, 3, R)$".

   Operations:

   - READCOLOUR(S, x, y): Return the colours at position (x, y), i.e., the set $\{\, c \mid (x, y, c) \in S \,\}$.
   - WRITECOLOUR(S, x, y, c): Assign colour c to position (x, y), i.e., add the triple (x, y, c) to S. If position (x, y) already has colour c, then do nothing.
   - NEXTINROW(S, x, y): Return the position of the next coloured pixel that appears after (x, y) and in the same row as (x, y), i.e., return $(x, \min\{\, y' \mid y' > y$ and $(x, y', c) \in S$ for some $c \,\})$. Return $(0, 0)$ if no such coloured pixel exists. **Assumption on input values:** You can assume that a pixel at (x, y) **exists** in the PIXELSET.
   - NEXTINCOLUMN(S, x, y): Similar to NEXTINROW, return the position of the next coloured pixel that appears after (x, y) and in the same column as (x, y). **Assumption on input values:** You can assume that a pixel at (x, y) **exists** in the PIXELSET.
   - ROWEMPTY(S, x): Return whether Row x is empty, i.e., return TRUE if and only if there does not exist a triple (x, y, C) with the given x in S.
   - COLUMNEMPTY(S, y): Similar to ROWEMPTY, return whether Column y is empty.

   Requirements: All above operations must have worst-case runtime $\mathcal{O}(\log n)$, where n is the total number of coloured pixels in the PIXELSET S.

   Give a *detailed* description of how to use AVL trees to implement PIXELSETS. In particular, answer the following questions.

   (a) How many AVL trees are you using? What does each node correspond to? What information is stored in each node?

   (b) What are the keys that you use for sorting each of the AVL trees? For each AVL tree, define **carefully and precisely** how you compare two pixels positioned at (x, y) and (x', y').

(c) For each of the above operations, describe in detail how it works, and argue why it works correctly and why its worst-case runtime is $\mathcal{O}(\log n)$.

Hint: Try to make use of textbook algorithms for BST and AVL trees, and please do **not** repeat algorithms or runtime analyses from class or the textbook—just refer to known results as needed.

2. We want to an augment AVL tree so that, in addition to the usual dictionary operations Insert, Delete and Search, it also supports the operation Average, defined as follows: given a pointer to any node x of T, Average(x) returns the average key-value in the subtree rooted at node x (including x itself). The query should work in worst-case time $\Theta(1)$.

   (a) What extra information needs to be stored at each node?

   (b) Describe how to modify Insert to maintain this information, so that its worst-case running time is still $O(\log n)$. Briefly justify your answer.

   (c) Describe how to modify Delete to maintain this information, so that its worst-case running time is still $O(\log n)$. Briefly justify your answer.