Submit your solutions electronically via Crowdmark. The assignment will be marked out of 65 (there is also 6 bonus marks). Please read [http://www.cs.umanitoba.ca/~kamalis/winter19/infoCOMP3170.pdf](http://www.cs.umanitoba.ca/~kamalis/winter19/infoCOMP3170.pdf) for guidelines on academic integrity.

**Problem 1  Quick-Select [6 marks]**

When doing Quick-Select and Quick-Select, it is desired to have a *good* pivot which is almost in the middle of the sorted array. When doing the average-case analysis of Quick-Select, we considered a *good* and a *bad* case; the good case happened when the pivot was among the half middle items of the sorted array, i.e., we had $n/4 \leq i < 3n/4$ ($i$ is the index of pivot in the partitioned array). In our analysis, we provided an upper bound for the time complexity of the algorithm in the *good* case and showed that $T(n) \leq T(3n/4) + cn$ in these cases for some constant $c$. Since the good case happened with probability 1/2, we could prove that the algorithm runs in linear time on average (see the recursion slide 10 of lectures on selections).

Change the definition of the good case and assume the good case happens when we have $n/10 \leq i < 9n/10$. Provide an upper bound for $T(n)$ and use that to show that Quick-Select runs in $O(n)$.

**Hint:** start by calculating the probability of good case and bad case happening.
Problem 2  Median-of-Three Algorithm [6+6+6=18 marks]

Consider a generalization of Median-of-Five algorithm which has a parameter $\alpha$ for an integer $\alpha \geq 1$. Instead of partitioning input into $n/5$ blocks of size 5, the algorithm partitions the input into $n/(2\alpha + 1)$ blocks of size $2\alpha + 1$ (assume $n$ is a power of $2\alpha + 1$). Note that the algorithm becomes the median-of-five algorithm when $\alpha = 2$.

a) Follow the same steps as slide 14 of lecture notes to derive a recursive formula for the time complexity $T(n)$ of this algorithm as a function of $n$ and $\alpha$ (there is no need to solve the recursion; just deduce the recursive definition of $T(n)$).

b) Assume $\alpha = 3$ (the algorithm will be “median of 7”). Rewrite the recursion for this particular $\alpha$ and try to solve the recursion by guessing that $T(n) \in O(n)$. Follow the same steps as in the slides and indicate whether we can state $T(n) \in O(n)$.

c) [bonus] Assume $\alpha = 1$ (the algorithm will be “median of 3”). Rewrite the recursion for this particular $\alpha$ and solve the recursion to provide a tight bound (in terms of $\Theta$) for the time complexity of this algorithm.

Problem 3  AVL Trees [6+6+6+8=26 marks]

This problem will concern operations on the AVL tree $T$ shown in the figure below.

![AVL Tree Diagram]

a) Show that $T$ is an AVL tree by writing in the balance at each node.

b) Draw the tree after performing operation $\text{insert}(4)$. Indicate any rotations that are required at each step.

c) Draw the original tree after performing operation $\text{delete}(8)$. Swap with its predecessor. It suffices to just draw the final tree.

d) Draw the original tree after performing operation $\text{delete}(8)$. This time, swap with its successor. It suffices to just draw the final tree.
Problem 4  (More) Binary Search Trees [5+8+8=21 marks]

a) Given two binary trees $T_1$ and $T_2$, each including $n$ distinct keys, describe an efficient algorithm that detects whether the two trees include the same keys. The algorithm returns ‘yes’ if any key in $T_1$ is also present in $T_2$ and vice versa, and returns ‘no’ otherwise. Your algorithm should run in $O(n)$. You need to describe your algorithm in a few English sentences and provide (a short) evidence that it runs in $O(n)$.

b) We define foo trees as follows. A foo tree is a binary search tree where for every node, the heights of the left and right subtree differ by at most 10. Prove that a foo tree with $n$ nodes has height $O(\log n)$.

c) Describe an efficient algorithm for computing the height of a given AVL tree. Your algorithm should run in time $O(\log n)$ on an AVL tree of size $n$. In the pseudocode, use the following terminology: $T\.left$, $T\.right$, and $T\.parent$ indicate the left child, right child, and parent of a node $T$ and $T\.balance$ indicates its balance factor (-1, 0, or 1). For example if $T$ is the root we have $T\.parent=nil$ and if $T$ is a leaf we have $T\.left$ and $T\.right$ equal to nil. The input is the root of the AVL tree.