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.

**Answer:**

We showed in the class that for the average cost of selection algorithm on an array of size $n$, we have

$$T(n) \leq cn + \frac{1}{n} \left( \sum_{j=0}^{i-1} T(n - j - 1) + \sum_{j=i+1}^{n-1} T(j) \right)$$

Assuming $n/10 \leq j < 9n/10$ (when pivot is good), we will have $n - j - 1 < T(9n/10)$ and $j - 1 < 9n/10$. Consequently, $T(n - j - 1) < T(9n/10)$ and $T(j - 1) < T(9n/10)$. Note that $9n/10 - n/10 = 4n/5$, i.e., with probability 4/5, the pivot is good and with probability 1/5, it is bad. From the above equation, we get:
\[ T(n) < cn + \frac{4}{5} \cdot T(\frac{9n}{10}) + \frac{1}{5} \cdot T(n) \implies \]
\[ T(n) < \frac{5}{4}cn + T(\frac{9n}{10}) \implies \]
\[ T(n) < \frac{5}{4}cn + \frac{5}{4} \times \frac{9}{10}cn + \frac{5}{4} \times \frac{81}{100}cn + \ldots + d \implies \]
\[ T(n) < d + \frac{5}{4}cn \sum_{i=0}^{\infty} \left(\frac{9}{10}\right)^i \in O(n) \]

**Marking Scheme:** 4 marks for showing the correct recursion and 2 marks for solving the recursion correctly.
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 $T(n)$). Answer: Assume $x$ is the selected median-of-medians. So, half of blocks have their medians smaller than $x$; each of these blocks have $\alpha$ element smaller than their median (and hence smaller than $x$). So, there are $\alpha + 1$ elements smaller than the median in each block. In total, there will be at least $\frac{1}{2} \cdot \frac{n}{2\alpha + 1} \times (\alpha + 1) = \frac{n(\alpha+1)}{4\alpha+2}$ items smaller than $x$. Similarly, there are at least $\frac{n(\alpha+1)}{4\alpha+2}$ items larger than $x$. Consequently, the size of recursion can be at most $n - \frac{n(\alpha+1)}{4\alpha+2} = \frac{3\alpha+1}{4\alpha+2} n$. Assume $T(1) = d$. For $n > 1$, we can write

$$T(n) \leq T\left(\frac{n}{2\alpha + 1}\right) + \frac{cn}{2\alpha + 1} + T\left(\frac{3\alpha + 1}{4\alpha + 2} n\right)$$

Marking Scheme: 3 marks for correct reasoning and inclusion of $T(2n/3)$; 3 marks for other elements of recursion.

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 $T(n) \in O(n)$. Answer: Let’s guess $T(n) \in O(n)$ and use strong induction to prove it. We should prove there is a value $M$ s.t. $T(n) \leq Mn$ for all $n \geq 1$. For the base we have $T(1) = d \leq M$ as long as $M \geq d$. For any value of $n$ we can state:

$$T(n) \leq T\left(\frac{n}{7}\right) + T\left(\frac{5n}{7}\right) + cn$$

from above recursion

$$\leq M \cdot \frac{n}{7} + M \cdot \frac{5n}{7} + cn = M \frac{6}{7} n + cn$$

induction hypothesis

As long as we have $M > 7c$, from the above inequality we can conclude $T(n) < Mn$. So, 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. Answer: First, let’s guess $T(n) \in O(n)$
and see if we can use strong induction to prove it. We should prove there is a value $M$ s.t. $T(n) \leq Mn$ for all $n \geq 1$. For the base we have $T(1) = d \leq M$ as long as $M \geq d$. For any value of $n$ we can state:

$T(n) \leq T(n/3) + T(2n/3) + cn$ (from above recursion)

$\leq M \cdot n/3 + M \cdot 2n/3 + cn$ (induction hypothesis)

$= (M + c)n$

Note that we cannot sow that $(M + c) \leq M$ for any value of $M$. So, following the same steps does Not give us the same result, i.e., we could Not prove that $T(n) \in O(n)$.

This algorithm’s complexity is in fact $O(n \log n)$; To prove it, we guess $T(n) < Mn \log n$ for some $M$ and use a strong induction.

$T(n) \leq T(n/3) + T(2n/3) + cn$ (from above recursion)

$\leq M \cdot (n/3)(\log n/3) + M \cdot (2n/3)(\log 2n/3) + cn$ (induction hypothesis)

$\leq (Mn/3)(\log n/3 + 2 \log (2n/3)) + cn = (Mn/3)(\log (4n^3/27)) + cn$

$= (Mn/3)(2 + 3 \log n - \log 27) + cn < (Mn/3)(3 \log n - 2) + cn = Mn \log n - 2Mn/3 + cn$

The above is less than $Mn \log n$ as long as $M < 2c/3$.

Marking Scheme: At most 2 mark if a proof was claim. Otherwise, 6 marks for getting to $(M + c) \leq M$; partial marks $\leq 4$ if failing in the proof on some other part.

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.

a) Show that $T$ is an AVL tree by writing in the balance at each node. Answer: See the figure below.

b) Draw the tree after performing operation insert(4). Indicate any rotations that are required at each step. Answer: Left-left scenario; a right rotation is applied. See the figure below.

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

d) Draw the original tree after performing operation delete(8). This time, swap with its successor. It suffices to just draw the final tree.

Answer:
Problem 4  (More) Binary Search Trees [5+8+8=21 marks]

a) Given two binary search 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 $O(n)$.

**Answer:** Create two sorted arrays $a_1$ and $a_2$ including the keys in the two subtrees. It takes $O(n)$ using in-order traversal. Then we can do a linear scan on the two sorted arrays to indicate whether they include the same set of keys.

**Marking Scheme:** 3 marks for mentioning the sorted array (by “in-order” traversal). 2 marks for mentioning the linear scan and a time complexity.

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 $O(\log n)$.

**Answer:** Let $N(h)$ denote the number of
nodes in a foo tree of height $h$. We can write $N(0) = 1$ and $N(-1) = 0$. For $h > 1$, we have $N(h) \geq N(h - 1) + N(h - 11)$; this is because one of the subtrees of a tree with height $h$ has height $h - 1$ and the other one can have height at least $h - 11$ by the definition of foo trees. Following the inequality, we get $N(h) \geq 2N(h - 11)$, and iterating $h/11$ times, we get $N(h) \geq 2^{h/11}N(0) = 2^{h/11}$. In other words, $\log(N(h)) = h/11$ or $h \leq 11\log(N(h))$. This means that a foo tree of $n$ nodes has height $O(\log n)$.

**Marking Scheme:** 5 marks for deducing the right recursion (including a brief justification) and 3 marks for solving the recursion.

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. **Answer:** Here is the algorithm:

```plaintext
height(T)
T: the root of an AVL tree
1. if T.left = nil then
2. return T.balance
3. else
4. return 1 + height(T.left) + \lceil T.balance/2 \rceil
```

If $T$.left = nil then $T$.balance is either 0 or 1, the height of the tree. This shows that the correct answer is always returned at line 2. Now use induction on $h$, the height of the tree. For the base case $h = 0$, we have $T$.left = nil, and we have already noted that the correct answer will be returned at line 2. Assume, to apply the principle of strong mathematical induction, that the algorithm is correct for all trees of height up to $h - 1$, some $h > 0$, and let $T$ be a tree of height $h$. If $T$.balance $\leq 0$ then $h$ is equal to 1 plus the height of $T$.left. If $T$.balance $= 1$ then $h$ is equal to 2 plus the height of $T$.left. In both of these cases the correct answer is returned at line 4 since $\lceil -1/2 \rceil = 0$, $\lceil 0/2 \rceil = 0$ and $\lceil 1/2 \rceil = 1$.

The nonrecursive work done during each call to height is $O(1)$, and the number of recursive calls is bounded by $h$. Since $h = O(\log n)$ the overall running time is $O(\log n)$.

**Marking Scheme:** 6 marks for the right algorithm (note that the $\lceil T$.balance/2 $\rceil$ can be replaced by a couple of if statements). 2 algorithms for a brief justification;
that includes a brief explanation on why the algorithm is correct and why it runs in $O(\log n)$. [no formal proof like above is necessary]