Homework 3

CS41, Spring 2023 — Alina Palacios and Nina Zhuo

1. *Proof.* We prove by induction that the number of leaves ℓ in a binary tree is always one more than the number of full nodes n, such that $\ell = n+1$. In a binary tree, a full node is any node with two children.

   i. **Base Case:** Given a binary tree with one leaf such that $\ell = 1$ and each node has at most one child, there are no full nodes. Adding an additional leaf to a parent node with one child will give us one full node and an additional leaf, so base case would not apply because $\ell \neq 1$. Adding an additional leaf to an existing leaf does not change the number of leaves or number of full nodes. Therefore, when $\ell = 1$, $n = 0$ such that the number of leaves is one more than the number of full nodes.

   ii. **Inductive Hypothesis:** We establish in our base case that the number of leaves ℓ is one more than the number of full nodes n when $\ell = k$. We want to prove that this holds true when we have $k+1$ number of leaves such that the number of full nodes will be one less than $k+1$.

   iii. **Inductive Step:** Suppose we have a binary tree with $k+1$ number of leaves. We remove a leaf from this tree such that the number of leaves is one less than $k+1$

   $$(k+1) - 1 = k$$

   In the case that the parent node of the removed leaf was a full node, the number of full nodes is now one less than before because the parent node now has one child instead of two. Thus, the number of full nodes is now

   $$(k) - 1 = k - 1$$

   Since the number of leaves k is one more than the number of full nodes $k-1$, our inductive hypothesis holds. Adding back that leaf restores the tree to $k+1$ leaves and k full nodes such that the number of leaves will always be one more than the number of full nodes.

   In the case that parent of the removed leaf is not a full node such that removing that leaf does not reduce the total number of leaves in the tree because the parent node becomes a leaf, we recursively remove a leaf from the tree until the parent of the removed leaf is a full node or we reach the root node, which we know works with our inductive hypothesis by the base case.

   □

2. If we imagine array A as sorted, the element that occurs k times will appear as $\frac{n}{3}$ consecutive indices. Thus, it is a guarantee that the repeated element will occur at either the $\frac{n}{3}$th, $\frac{2n}{3}$th, or nth indices because the block of elements that occur k times in the sorted array will take up some third of the array.

   We know that D-select will find the i-th element in an unsorted array. Therefore, we can find our candidate elements by calling D-select three times for the $\frac{n}{3}$th, $\frac{2n}{3}$th, and nth elements

in the sorted version of the array. Looping over the length of unsorted A to check which elements match any of our three candidate elements allows us to find the element that occurs k times in linear time.

Each iteration of D-selection takes $O(n)$ time and checking which elements in A match any of our three candidate elements also takes $O(n)$, so

$$3 \cdot O(n) + O(n) = 4 \cdot O(n)$$

giving us overall running time of $O(n)$.

3. The first step of the algorithm is start by examining the middle column and middle row of the grid. Find the smallest element among the middle column and middle row and start there. Next, from the position where we found the smallest element, check if there is a smaller element the neighboring elements within the adjacent quadrants formed by the middle column and middle row.

   Case 1: If the current element is smaller than all of the adjacent elements, then we stop and have found the local minimum.

   Case 2: If there is an element smaller than our current element, we move to the quadrant that element is in. By doing this, we have cut the grid by $n/4$ and can look for the local minimum in one quadrant. We will query for smaller neighboring elements recursively until we find an element that is less than or equal to all of its neighbors.

   We can show the time complexity of our algorithm using the Master Theorem. Our recurrence relation is:

$$T(n) \leq T(\tfrac{n}{4}) + O(n)$$
$$a = 1, b = 4, c = 1$$

$$\frac{1}{4^1} = \frac{1}{4}$$

   Since $\frac{1}{4} < 1$, we can use the $O(n^c)$ portion of the Master Theorem which is $O(n^1)$. We can conclude that the running time is $O(n)$.