CS41 Homework 8

This homework is due at 11:59PM on Sunday, November 4. Write your solution using LaTeX. Submit this homework using **github** as a **.tex** file. This is a **partnered homework**. You should primarily be discussing problems with your homework partner.

It's ok to discuss approaches at a high level with others. However, you should not reveal specific details of a solution, nor should you show your written solution to anyone else. The only exception to this rule is work you've done with a lab partner *while in lab*. In this case, note (in your **README** file) who you've worked with and what parts were solved during lab.

1. **Find the missing integer** (CLRS 4-2) Suppose $n = 2^k - 1$ for some k.

   An array $A[1 \ldots n]$ contains all the integers from 0 to n except one. Each integer from 0 to n is represented as a k-bit string. It would be easy to determine the missing integer in $O(n)$ time by using an auxiliary array $B[0 \ldots n]$ to record which numbers appear in A. Unfortuantely, we cannot access an entire integer in A with a single operation. Because the elements of A are represented in binary, the only operation we can use to access them is "fetch the j^{th} bit of $A[i]$", which takes constant time. This means that reading every digit of every number in A would take $O(nk) = O(n \log n)$ operations.

   In this problem, we'll develop an efficient divide and conquer algorithm that identifies the missing integer, using only $O(n)$ operations.

   (a) If one number x is missing, it must be the case that either $x < n/2$ or $x \geq n/2$. Describe how to figure out which of these is true using only $O(n)$ operations.

   (b) After you figure out whether $x < n/2$ or $x \geq n/2$, which bit(s) of x do you know?

   (c) Define the sets:

   $$A_{\text{small}} = \{y \in A \mid y < n/2\}$$
   $$A_{\text{big}} = \{y \in A \mid y \geq n/2\}$$

   We'd like to use the insight from part (1a) to intelligently decide which elements to put in A_{big} and which to put in A_{small}. This will be our preprocessing step to set up the "divide" part of our divide and conquer algorithm. Describe a way to keep track of which entries of A belong to either A_{small} and A_{big}, using only $O(n)$ work.

   (d) Put together the two parts above into an algorithm that recurses on either A_{small} or A_{big}. Part (1c) should help you determine your "divide" step, and part (1b) should help you determine how to "combine" the recursive return value with new information to figure out x.

   Describe your algorithm with low-level pseudocode.

   (e) Write a recurrence for the runtime of this algorithm and solve it using partial substitution.

2. **Counting significant inversions** (K&T 5.2)

   Recall the problem of finding the number of inversions between two rankings. As we saw, we are given a sequence of n numbers $a_1, a_2, \ldots, a_n$, which we assume are all distinct, and we define an inversion to be a pair of indices $i < j$ such that $a_i > a_j$.

   We previously used counting inversions as a good measure of how different two orderings are. However, one might feel that this measure is too sensitive. Let's call a pair a *significant inversion* if $i < j$ and $a_i > 2a_j$. Give an $O(n \log n)$ algorithm to count the number of significant inversions.

3. **Database queries** (K&T 5.1)

   You are interested in analyzing some hard-to-obtain data from two separate databases. Each database contains n numerical values (so there are $2n$ values total). You'd like to determine the *median* of this set of $2n$ values, defined as the n-th smallest value.

The only way you can access these values is through *queries* to the databases. In a single query, you can specify a value k to one of the two databases, and the chosen database will return the k-th smallest value it contains. Since queries are expensive, you would like to compute the median using as few queries as possible.

(a) Design an algorithm that finds the median value using at most $O(\log n)$ queries. Full pseudocode is not necessary, but you must clearly explain how it works, and you must handle all edge cases (e.g., do not assume that n is even).

(b) Prove that your algorithm correctly returns the median.

(c) Prove that your algorithm uses only $O(\log n)$ queries.

4. **(extra challenge) Divide and conquer for minimum spanning trees (V2.0)**

In lab, we considered a divide-and-conquer approach to the minimum spanning tree problem, with the rough outline that the algorithm:

- Divides the graph into two pieces, $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$. ($V_1 \cup V_2 = V$ and V_1 and V_2 are disjoint. $|V_1|$ and $|V_2|$ are each roughly half of $|V|$. E_1 is the edges in E with both endpoints in V_1, and E_2 is the edges in E with both endpoints in V_2.)
- Recursively finds the MSTs M_1 for G_1 and M_2 for G_2.
- Finds the lowest-weight edge $e = (u, v)$ with $u \in V_1$ and $v \in V_2$.
- Returns the minimum spanning tree $M_1 \cup M_2 \cup \{e\}$.

In lab, your group came up with an example weighted, connected input graph G and a particular execution so that the algorithm did not return a minimum spanning tree of G.

Is it possible to "patch" this algorithm to work, if the vertex partition is chosen cleverly? That is, can we do a little bit of conquering *before* the divide step(s), which will make this divide-and-conquer MST algorithm work?

If YES, then describe how to fix this divide and conquer algorithm to be correct. If NO, then argue why no rule for dividing G can make the algorithm correct.