CS41 Homework 8

This homework is due at 11:59PM on Sunday, November 12. Write your solution using LaTeX. Submit this homework using **github** as a **.tex** file; the code should be in a file called **palindrome.py**. 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 teammate *while in lab*. In this case, note (in your **homework feedback poll**) 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. Unfortunately, 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. That is, you should specify the data structures that you are using, the indices of your loops, the types of your variables, how you access data, etc., so that anyone reading your algorithm would be able to implement it exactly according to your description.

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

2. **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.

3. **Longest Palindrome.**

   Let Σ be a finite set called an *alphabet*.[1] A *palindrome* is a string which reads the same backwards and forwards. Let s be a string of characters from Σ and let $x \in \Sigma$ be some character. The reversal of s is denoted s^R. Then the strings ss^R (that is, s concatenated with s^R) and scs^R are both palindromes.

   In the **Longest Palindrome Problem**, you're given a string s of n characters from Σ and must output the length of the longest palindrome that is a substring of s.

   (a) *Briefly* describe a simple $\Theta(n^3)$ algorithm that solves the longest palindrome problem. Why is your algorithm $\Theta(n^3)$?

   (b) Design an algorithm that uses dynamic programming to solve the longest palindrome problem in less than n^3 time. (You *must* use dynamic programming, even if you have a fast idea which does not use dynamic programming!)

   (c) Modify your algorithm so that it also returns the longest palindrome in x (and not just its length).

   (d) Code your dynamic programming algorithm using tabulation in the file **palindrome.py**. Some example runs:

```
$ python3 palindrome-soln.py "racecars"
The length of the longest palindrome is 7.
The longest palindrome is: racecar
$ python3 palindrome-soln.py "quick brown racecars jump over the sleepy fox"
The length of the longest palindrome is 7.
The longest palindrome is: racecar
$ python3 palindrome-soln.py "a quick brown racecar never jumps over the sleepy fox"
The length of the longest palindrome is 11.
The longest palindrome is: n racecar n
python3 palindrome-soln.py "abcdeedba"
```

[1] For example, Σ might be $\{0, 1\}$ or $\{a, b, c, \ldots, z\}$.

```
The length of the longest palindrome is 4.
The longest palindrome is: deed
$ python3 palindrome-soln.py "CS35: data structures and algorithms"
The length of the longest palindrome is 3.
The longest palindrome is: ata
$ python3 palindrome-soln.py "a man a plan a canal Panama"
The length of the longest palindrome is 3.
The longest palindrome is: ama
$ python3 palindrome-soln.py "amanaplanacanalpanama"
The length of the longest palindrome is 21.
The longest palindrome is: amanaplanacanalpanama
```

Note that you do not need to remove whitespace or fix the cases of letters.

You should feel free to run your own tests, and you should make sure that you have debugged your code (and your dynamic programming algorithm!). Your code should be a straightforward tabulation implementation of your dynamic programming plan.

4. **(extra challenge) Cassie's Convenience Stores (v2.0)** It is natural to assume the profit of a convenience store changes depending on how close to other convenience stores it is. Suppose instead of an array of profits $P[1 \cdots n]$ and a minimum distance between stores K, Cassie does more market research and gets information on $P[k, d]$, the annual profit of location k assuming that the closest other store is at least d km away.

   Design an algorithm that computes Cassie's maximum profit.