Analysis of Algorithms
CS 375, Spring 2019
Homework 2
Due **AT THE BEGINNING OF CLASS** Monday, February 18

- From your textbook (CLRS), please read Chapters 1 and 2.
- Unless otherwise specified, exercises will be from the CLRS textbook and will be named on HW assignments by exercise number used in the book.
- *A general note:* When writing up your homework, please write neatly and **explain your answers clearly**, giving all details needed to make your answers easy to understand. Graders may not award credit to incomplete or illegible solutions. Clear communication *is* the point, on every assignment.

Exercises

1. CLRS 2.1-1 (pg. 22).
2. (This question is moved from HW1.) Design a *recursive* algorithm to find all the common elements in two sorted lists of numbers. For example, for input lists $[2, 5, 5, 5]$ and $[2, 2, 3, 5, 5, 7]$, the output should be the list $[2, 5, 5]$. What is the maximum number of comparisons your algorithm makes if the lengths of the two input lists are m and n, respectively?

   Please give both a pseudocode description and an English description, to make it as easy as possible to understand the algorithm, and explain how you know it solves the problem correctly.

   Some **hints** and comments for you:

   - There are multiple possible solutions for this exercise! One possible way to approach it would be to "divide" a list for a recursive algorithm similarly to how we divided the list / array when we designed Selection Sort in class—that is, for a list L, have a recursive call be on all but the first element of L. The algorithm will do some processing that includes the first element of L, however, as part of the "combine" / "conquer" steps.

     To think about as part of designing the algorithm: What if the first element of L were one of the elements to be returned in the common-elements list? What if it weren't? How might your algorithm handle these two cases?
   - This algorithm takes two lists as input, but it can be useful to think of a base case and a recursive divide-and-conquer structure in terms of one of them. For example, a good base case for lists would be the empty list. What would the algorithm return if the first input list were empty?

     How could you design the divide step so that each recursive call brought the first list closer to that base case (if it weren't already the empty list!)?

Under what conditions would the algorithm also bring the second list closer to the base case? And under what conditions would it not do so?

Would it be useful to also have a base case for the case where the second list is empty?

- For this exercise, please don't worry about optimizing the algorithm for efficiency! The objective is to practice thinking about recursive design, and correctness is the primary goal.
- It's fine if your solution doesn't follow the structure suggested in these hints!

3. Here are a few exercises on properties of logarithms: You'll be asked to show that a few properties of logarithms are true. The objective is to help you become more familiar with some properties of logarithms that are especially important for analysis of recursive algorithms.

   As an example, here's how we might show that $\log_b xy = \log_b x + \log_b y$. In the below, recall from the definition of logarithm that a logarithm really stands for an exponent—$x = log_b n$ means that $b^x = n$, i.e., x is the exponent we raise b to, to get n.

   > To make it easier to talk about raising some relevant expressions to a power, we'll introduce a few variables: let $k = \log_b xy$, $\ell = \log_b x$, and $m = \log_b y$. Then, by the definition of logarithm (above), $b^k = xy$, $b^\ell = x$, and $b^m = y$. (Note: Do you see why?) Therefore, $b^k = b^\ell b^m = b^{\ell+m}$, and thus, by standard properties of exponents, $k = \ell + m$, which is what we had set out to show.

   Now, pick 2 of the following 3 properties of logarithms (your choice!) and show that they are true. (If you turn in all 3 of them, only your best 2 will be counted for your grade, so I hope you'll give all 3 of them a try!)

   Please give detailed explanations, as shown in the example above! Assume that a, b, c, n are positive real numbers (but note—they are not necessarily integers).

   (a) $\log_b a^n = n \log_b a$.

   (b) $\log_b n = \frac{\log_c n}{\log_c b}$. (This shows that changing the base of a logarithm—say, from $log_b n$ to $log_c n$—really means multiplying by a constant factor, because this could be rewritten as $log_b n = \frac{1}{log_c b} \cdot log_c n$.)

   (c) $a^{\log_b n} = n^{\log_b a}$. (This result is sometimes called the "log-switching theorem" since it says that we can "switch" a and n.)