

Analysis of Algorithms
CS 375, Spring 2020

Homework 5

Due **AT THE BEGINNING OF CLASS** Wednesday, February 26

- Please recall the guidelines for algorithms given on previous HW assignment sheets. They continue to apply to all HWs in CS375.
- In general, there may be multiple correct ways of presenting an algorithm, although excessively inefficient or inelegant solutions may not receive full credit. If you have questions about whether your proposed solution is excessively inefficient or inelegant, please ask your Prof.!
- *A general note for CS375:* 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 full credit to incomplete or illegible solutions. Clear communication *is* the point, on every assignment.

In general in CS375, unless explicitly specified otherwise, answers should be accompanied by explanations. Answers without explanations may not receive full credit. Please feel free to ask me any questions about explanations that might come up!

Exercises

1. **(Common elements with HW1!)** In contrast to the iterative algorithm you created for HW1, here, design a *recursive* algorithm to find all the common elements in two sorted LLists of numbers. (Please be sure to use the LList data structure from class!) 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 between list elements 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 give an explanation of your algorithm's correctness. (As usual for these LList exercises, the only functions you can use as primitives are the three given in the definition, and the check if a list is empty. You must write any others yourself for this exercise, along with correctness arguments. This does not, however, suggest that any are necessary to solve this problem!)

2. **No cabin required!** Sometimes in complexity analysis and other mathematics, we may use a function that, on input n that could be any real number (not necessarily an integer), returns the greatest integer less than or equal to n . This function is typically called the *floor* function, and here are some examples of its use: $\text{floor}(42) = 42$; $\text{floor}(3.14159) = 3$, $\text{floor}(3.78) = 3$.

For this exercise, you do not need to write a floor function! Instead, write a straightforward recursive function *log_2_Floor* that combines the floor function with taking the base-2 logarithm of an input:

Input: Real number $n \geq 1$ (does not need to be an integer!)
Output: The greatest integer less than or equal to $\log_2 n$

As examples of how *log_2_Floor* works:

[illegible]

(It's worth noticing how slowly the log function grows as input gets really, really big! Even for very large n , $\log_2 n$ does not get very large. This will be an important idea in our complexity analysis later this semester!)

Your solution must be recursive. **In addition, your solution can only use the basic arithmetic operations $+$, $-$, $*$, $/$ and $\%$** when performing number-valued arithmetic (i.e., non-boolean) operations. (The modulo operator $\%$ can be thought of as returning the *remainder* of a division—for examples, $5 \% 2 = 1$ and $7 \% 4 = 3$, because the remainder when dividing 5 by 2 is 1, and the remainder when dividing 7 by 4 is 3.) In particular, you may not use a floor function in your solution.

As usual, give a short English explanation of correctness; because the algorithm is recursive, make sure it's an inductive explanation.