

Analysis of Algorithms
CS 375, Fall 2019

Homework 5

Due **AT THE BEGINNING OF CLASS** Wednesday, September 25

- 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:* 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. **(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 *log2-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 $\log_2 n$

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 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.)

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

3. Write a recursive algorithm that takes a natural number n as input, returns nothing, and prints number n one line at a time on the screen. For example, on input 375, the output on the screen would be

3
7
5

In your algorithm, **for full credit, you may not use any string functions (other than print)**—the input is a number and should be treated that way. In addition, **for full credit, your algorithm should use only the basic arithmetic functions $+$, $-$, $*$, $/$, and the modulo operator `%`** when performing number-valued arithmetic operations. (That’s the same as in the previous exercise.)

Note: This function is different from others so far in CS375, in that it does not have a return value—instead, it simply generates output for the screen. The specification for it therefore includes *side effects*:

```
# Input: natural number n
# Output: none
# Side Effect: prints number n to screen, one digit per line
```

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