

Analysis of Algorithms
CS 375, Spring 2021

Problem Set: Divide & Conquer II

Due **BY THE BEGINNING OF CLASS** Monday, March 15

- From your textbook (CLRS), please read Chapters 2.1, 2.2, and 2.3.1.
- To submit this Problem Set, the standard file naming and HW submission conventions for CS375 apply: Please put your answers in a PDF file named

`CS375_PSDivConqII_<userid>.pdf`

where `<userid>` is replaced by your full Colby userid (including your class year)—for example, my file would be called `CS375_PSDivConqII_eaaron.pdf`—and submit it to your `SubmittedWork` folder. (Please note the II—not I—in the filename).

- Unless otherwise specified, exercises will be from the CLRS textbook and will be named on HW assignments by exercise number used in the book.
- **IMPORTANT:** Some of these exercises may build upon topics covered in our Mar. 3 class meeting; they are included here “early” so you can see all of the exercises on this assignment.
- Exercises on this problem set use the *LList* data structure as defined in our lecture notes. As with all LList exercises in CS375, the only functions you can use as primitives are the three given in the definition (*first*, *rest*, *cons*), and the check if a list is empty. You must write any others yourself for these exercises, along with correctness arguments. This is not intended to suggest that any new or additional LList functions are necessary to solve any particular exercise; if you find them helpful in some cases, however, this is intended to clarify that you are welcome to create helper functions or other new functions for use on LLists to use as part of your solutions.
- 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:* As always, please present answers cleanly and **explain them clearly and thoroughly**, giving all details needed to make your answers easy to understand; typed-up (rather than handwritten) answers are especially appreciated. (Feel free to talk with your Prof. or TA’s about using L^AT_EX to typeset your answers!) 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. Using the *LList* data structure from class, write a recursive algorithm for the *reverse* problem on lists:

```
# Input: List  $L = [a_0, a_1, \dots, a_n]$ 
# Output: List  $L' = [a_n, \dots, a_1, a_0]$  with the same elements as in  $L$ 
#           but in reverse order
```

You may not use other data structures (including other kinds of lists) in your algorithm—this is an exercise on LLists! As usual, give a short English explanation of correctness; because the algorithm is recursive, make sure it’s an inductive explanation.

As usual, give a short English explanation of correctness for every algorithm given for this exercise—definitely *LLReverse*, but also any other algorithms you create for use here. For each recursive algorithm on LLists (and I anticipate any algorithm you write for this exercise would be recursive!), make sure the accompanying explanation is an inductive explanation.

2. **(Common elements with PS1!)** In contrast to the iterative algorithm you created for the first Problem Set of the semester, here, design a *recursive* algorithm to find all the common elements in two sorted LLists of integers. (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.

The next two exercises below refer to nested LLists—an LList can, in the usual way, have an LList as an element. For example, $[55, [77, 42], [11, 42], 88]$ is a valid LList. (Please note that LLists can contain non-integer data—these simple examples show only integers, but the definition of LList allows for other possibilities too!)

For terminology, we say that an element of an LList is *top-level* to distinguish it from elements of an LList nested inside another LList. For example, in LList $[55, [77, 42], [11, 42], 88]$, 55 is top-level, while 77, 11, and both instances of 42 are not top-level. (Please talk with your Prof. if there are any questions about this definition!)

3. Using the LList data structure, write a recursive algorithm for the *LLRemoveAll* problem on lists:

```
# Input: Item  $x$  (assume  $x$  is not a list) and LList  $L$ 
# Output: List  $L'$  containing exactly the elements of  $L$ 
#         not equal to  $x$ , in the order in which they occur in  $L$ 
#         (see examples below)
```

(**Please note:** This specification requires that x is not a list—it is a single, non-list element!)

This function removes all—and only!—top-level occurrences of x in L . For examples,

- `LLRemoveAll(42, [55, 77, 42, 11, 42, 88])` returns `[55, 77, 11, 88]`
- `LLRemoveAll(42, [55, [77, 42], [11, 42], 88])` returns `[55, [77, 42], [11, 42], 88]`. (Note that 42 is not top-level!)
- `LLRemoveAll(42, [42, 67, 42, [42, 42, 43], 47])` returns `[67, [42, 42, 43], 47]`. (Only the top-level 42's are removed!)

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

4. Unlike the *LLRemoveAll* function above, which only removes an item that is top-level in an LList, an *LLDeepRemoveAll* function removes an item at any level of nesting. Using the LList data structure, write a recursive algorithm for the *LLDeepRemoveAll* problem on lists:

```
# Input: Item  $x$  (assume  $x$  is not a list) and LList  $L$ 
# Output: List  $L'$  with the same structure and elements as  $L$ 
#         (in that order) except all occurrences of  $x$  are removed
#         at any level of nesting in  $L$  (see examples below)
```

(**Please note:** Once again, this specification requires that x is not a list—it is a single, non-list element!)

For examples,

- `LLDeepRemoveAll(42, [42, 67, 42, [41, 42, 43], 47])` returns `[67, [41, 43], 47]`.
- `LLDeepRemoveAll(42, [55, [77, 42], [11, 42], 88])` returns `[55, [77], [11], 88]`.
- `LLDeepRemoveAll(47, [42, 47, [1, 2, [47, 48, 49], 50, 47, 51], 52])` returns `[42, [1, 2, [48, 49], 50, 51], 52]`.
- `LLDeepRemoveAll(3, [3, [[[3]]], [3, 4]])` returns `[[[[ ]], [4] ]]`.

For this exercise, you will also need to test whether or not an element in a list is a list itself. For this, you can use the following (or something very similar) in your pseudocode to test whether or not a list element e is itself a list:

```
if type(e) == list
    # if e is a list...
else
    # if e is not a list...
```

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