

Algorithm Design and Analysis
CS 376, Spring 2023

Problem Set 0 — **REVISED**

Due **AT THE BEGINNING OF CLASS** Monday, February 20

- Standard file naming and PS submission conventions apply: Please submit your type-written answers in a PDF file named

CS376_PS0_<userid>.pdf

where <userid> is replaced by your Colby userid—for example, my file would be called CS376_PS0_eaaron.pdf. **Please be sure to give your full userid, including the numbers representing your class year.**

Submit your PDF file by placing it in your `Private` folder in your workspace for CS376 on `filer`. **As always, please be sure it is in your *Private* folder.** If for any reason `filer` is inaccessible to you, emailing your PDF to me by the deadline is acceptable, but submitting to `filer` is the expected way of submitting Problem Sets and Smaller Assignments in CS376 this semester.

- From your textbook (CLRS), please read:
 - Appendices A, B.1–B.2 (that’s Appendix B, sections 1 through 2), C.1, D.1. (The rest of Appendix B will be helpful later in the course, too!)
 - Chapters 1 and 2.1–3.
- A few essential style guidelines for writing algorithms in CS376:
 - Unless explicitly instructed otherwise, avoid using `break` (or `continue` or `pass`) statements in algorithms for CS376—employing them can interfere with common approaches to reasoning about code correctness (e.g., using loop invariants). Instead, please find other approaches, perhaps using boolean variables to indicate loop termination conditions.

Answers employing `break` (`continue`, `pass`) may not receive full credit. Please feel free to talk with me about any particular situations that might come up—I’m happy to talk with you about how to avoid `breaks` in your code!
 - When presenting an algorithm, avoid using language-specific commands/routines (without comments) in pseudocode. Pseudocode, by definition, is supposed to be language-independent, at a level of abstraction higher-level than particular programming languages.
 - Please be sure to initialize variables or declare inputs/outputs, and be sure the purpose of **every variable**—including the names of functions or algorithms—can be quickly and fully understood by a reader; typically, this is done either by using descriptive variable names or by describing variables in English. Algorithms in which variables cannot be promptly understood may not receive full credit, so please feel free to ask me about variable names and descriptions—I’m happy to talk with you about particular situations that might come up!

- Often, the clearest way to present an algorithm is with both pseudocode and English description. Pseudocode must be accompanied by some kind of clarifying English description to count as fully explained.
- **NEW TO THE REVISED HW0:** Although there may be many correct ways to write an algorithm in pseudocode and explain it in English, excessively inefficient or needlessly complicated algorithms or explanations may not receive full credit. If you'd like to ask me about whether your algorithm and explanation seem needlessly complicated before you submit your work, please let me know!

Please note that, unless explicitly instructed otherwise, these guidelines apply to all exercises in CS376, whether or not they explicitly appear as part of the assignment itself. (This is similar to how all answers should be accompanied by explanations unless explicitly specified otherwise, whether or not that instruction is explicitly given as part of the assignment.)

- *A general note for CS376:* When writing up your homework, **explain your answers clearly**, giving all details needed to make your answers easy to understand. Clear communication *is* the point, on every assignment.

Unless explicitly specified otherwise, answers in CS376 should be accompanied by explanations; as part of this, in general, when you present an algorithm, give an accompanying correctness argument. Answers without convincing explanations that show command of the relevant concepts may not receive full credit. Please feel free to ask me any questions about explanations that might come up!

Exercises

1. Give proofs for the following properties of logarithms, assuming that a, b, c, n are positive real numbers. (These properties will be useful for complexity analysis later in CS376!) Write your proofs out carefully.
 - (a) $\log_b a^n = n \log_b a$.
 - (b) $\log_b a = \frac{\log_c a}{\log_c b}$.
 - (c) $a^{\log_b n} = n^{\log_b a}$.
2. CLRS B.2-3 (page 1165). Please write each relation explicitly as a finite set of ordered pairs over set A , where $A = \{1, 2, 3, 4\}$, and give an explanation for each answer that shows command of the relevant concepts, including relevant definitions and specific examples illustrating how the definitions apply or specific counterexamples showing how sets do not fit those definitions.

NOTE: Some 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.

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)
```

This removes all—and only!—top-level occurrences of x in L , i.e., those that are not nested inside other lists. 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, explain your answer and give a correctness argument.

4. CLRS 2.3-5 (page 39), modified a little: Write a recursive binary search, not iterative; and you do not have to give a complexity argument. (As usual, though, explain your answer and give a correctness argument.)
5. Use induction on n to show that for all integers $n \geq 0$

$$1 + a + a^2 + a^3 + \dots + a^n = \frac{a^{n+1} - 1}{a - 1}$$

where a is an arbitrary real number other than 1. Where does your proof rely on the fact that $a \neq 1$?

Give a fully detailed proof, as discussed in class and illustrated by sample proof documents at our *Additional Notes / Readings* webpage.

6. 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 376, the output on the screen would be

```
3
7
6
```

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** $+$, $-$, $*$, **integer division** $/$, **and the modulo operator** $\%$ when performing number-valued arithmetic operations, it should not use auxiliary functions, and it should not use other arithmetic functions (e.g., power or log) even if you write those functions yourself.

7. Unlike the *LLRemoveAll* function, 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  $x$  and LList  $L$ . Assume  $x$  is not a list.
# 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)
```

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 an item x is a list:

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

As usual, explain your answer and give a correctness argument.