

Algorithm Design and Analysis
CS 376, Colby College
Prof. Eric Aaron

OPTIONAL Additional Exercises with LList and IBT structures

This document presents a few **OPTIONAL** additional exercises with our LList and IBT data structures, as defined in lecture notes. Because those data structures are recursively defined, solutions to these exercises should be recursive algorithms; full solutions would also include inductive proofs of correctness—and, once we’ve covered *recurrences* in class, full solutions would also include statements of recurrences for algorithms’ complexities and solutions of those recurrences, resulting in an asymptotic complexity class (big-O, Θ , or Ω).

These exercises are not to be submitted—but I’m happy to go over them with you if you’d like! As always, please be in touch with questions.

Exercises

1. Using our *IntBinTree* data structure from class (*IBT*, for short), come up with a recursive algorithm that returns the number of integers in a tree.

Input: IntBinTree T
Output: The number of integers in tree T (0 if it’s empty)

Give an inductive explanation of the algorithm’s correctness.

2. Using our *LList* data structure from class (*IBT*, for short), come up with a recursive algorithm that returns the number of integers in an LList of integers.

Input: LList L of integers
Output: The number of integers in list L (0 if it’s empty)

Give an inductive explanation of the algorithm’s correctness.

3. Using our *IBT* data structure, come up with a recursive algorithm that returns the sum of the elements in a tree.

Input: IntBinTree T
Output: The sum of all of the integers in tree T

What did you decide that the algorithm should return on an empty tree as input? Explain your reasoning for that decision (a sentence or so could be sufficient), and give an inductive explanation of the algorithm’s correctness.

4. 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

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

5. **(Common elements!)** This is a two-part exercise, asking for both iterative and recursive solutions. The exercises have common elements, and they're about common elements!

- (a) Design an *iterative* (i.e., without using recursion) algorithm to find all the common elements in two sorted lists of numbers. **[For this non-recursive exercise, please do not restrict yourself to using LLists. See the next exercise, however]** For example, for input lists $[2, 5, 5, 5]$ and $[2, 2, 3, 5, 5, 7]$, the output should be the list $[2, 5, 5]$.

```
# Input: Two sorted lists of elements, S = [s_1, ..., s_m]
#         and T = [t_1, ..., t_n]
# Output: List of numbers L = [n_1, ..., n_k] where n_i is
#         a member of L exactly when it is an element of
#         both S and T. Also, for each value v_i that occurs
#         on L, the number of times it occurs on L is equal to
#         the minimum of the number of times v_i occurs on S
#         and the number of times v_i occurs in T.
#
#         For example, if S = [2, 5, 5, 5] and
#         T = [2, 2, 3, 5, 5, 7], the return value should be
#         the list [2, 5, 5].
```

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.

In addition, answer this question: What is the maximum number of comparisons your algorithm makes—i.e., number of times a pair of numbers is compared—if the lengths of the two input lists are m and n , respectively?

Also, please give a concise but convincing explanation of the most informative worst-case time complexity bound you can give of your algorithm, expressed with asymptotic complexity notation.

Note: Recall that there are different operations to add an element to a list (`append`, in Python) and to combine two lists into one (`extend`, in Python). If you use either or both in your answer, please make sure it is clear which operation is being used. You are also welcome to use other common list operations such as `insert` or `remove`, if you'd like.

- (b) In contrast to the iterative algorithm you created for the above exercise, 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!)