

Analysis of Algorithms
CS 375, Fall 2019

Homework 1

Due **AT THE BEGINNING OF CLASS** Wednesday, September 11

- From your textbook (CLRS), please read:
 - Appendices A.1 and B.1.
 - Chapters 1, 2.1, and 2.2.
- **IMPORTANT:** Some of these exercises may build upon topics covered in our Sept. 9 class meeting; they are included here “early” so you can see all of the exercises on this assignment.
- Unless otherwise specified, exercises will be from the CLRS textbook and will be named on HW assignments by exercise number used in the book.
- *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. **Summation exercises!** Familiarity with summations and $\sum$ notation is essential for algorithm complexity analysis! We’ll see more of that in a few weeks, but in the interest of being familiar with concepts before we need them, here are some warm-up exercises. For these exercises, as usual for CS375, please be sure to show your work and give a short but informative explanation for every answer.
 - (a) Exercise A.1.1 (pg. 1149). (A “simple formula” here means a mathematical expression with no $\sum$ notation, for which plugging in a value for n will give the numeric value of the formula. For example, a simple formula for $\sum_{i=1}^n i$ is $\frac{n(n+1)}{2}$.)
 - (b) Find a simple formula for $\sum_{i=1}^n 2(3i - 1)$.
2. **Set exercises!** Similarly, familiarity with sets and set operations is important for Computer Science in general—and for algorithm design and complexity analysis in particular—so here are some warm-up exercises. Give a short explanation for every answer—less than one sentence could suffice, as long as it demonstrates understanding of the relevant definition(s) / reasons for the answer.

For the exercises below, let $A = \{x, y, z\}$ and $B = \{x, y\}$.

- (a) True or False: $A \subseteq B$?
- (b) True or False: $B \subseteq A$?
- (c) True or False: $B \subseteq B$?
- (d) What is $A \cup B$?

- (e) What is $A \cap B$?
 - (f) What is $A \times B$?
 - (g) What is $\mathcal{P}(B)$, the power set of B ?
3. Design an *iterative* (i.e., without using recursion) algorithm to tell if one set is a *proper subset* of another.

```
# Input: Two lists of elements, S and T, representing sets
# Output: True if S is a proper subset of T, False otherwise
def properSubset(S,T):
    ...
```

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. (A complete explanation of correctness will almost certainly involve showing an understanding of the definition of proper subset.) **NOTE:** Needlessly inefficient algorithms, such as those that iterate over S or T more often than is needed for a straightforward solution, may not receive full credit!

For this, we'll assume that sets are represented by lists, but because sets cannot have repeated elements, the lists will be assumed to represent sets correctly—there are no repeated elements on either list.

4. Exercise 1.2.3 (pg. 14). (We need only consider natural number values of n .)
5. Design an *iterative* (i.e., without using recursion) algorithm to find all the common elements in two sorted lists of numbers. 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 your algorithm makes if the lengths of the two input lists are m and n , respectively?

```
# 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.

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. Of course, you are also welcome to use other common list operations such as `insert` or `remove`, if you'd like!

6. (A problem solving puzzle!) There are four people who want to cross a bridge, all of which begin on the same side. They have 17 minutes to get to the other side. As is common in these kinds of puzzles, however, there's a catch!

Because the bridge is old and weakened by time, a maximum of two people can cross the bridge at one time. Moreover, it's night time, and they have one—only one—flashlight. Any time people cross (whether one person or two people), they must have the flashlight with them, and the flashlight must be walked back and forth over the bridge (it can't, e.g., be thrown from one side to the other).

Person 1 takes 1 minute to cross the bridge, person 2 takes 2 minutes, person 3 takes 5 minutes, and person 4 takes 10 minutes. If a pair crosses the bridge together, they must walk together at the pace of the slower person.

Can all four of them get to the other side in 17 minutes? If so, how? If not, why not? Be sure to explain your answer!

(Note: A popular Algorithms textbook notes that, according to a rumor on the Internet, interviewers at a well-known software company located near Seattle have given this problem to interviewees!)

Important: For this exercise, *explain the full thought process by which you arrived at your answer!* Or, if you aren't able to find a full answer, explain the thought process as far as you get with your reasoning. This exercise is intended to give practice with thinking through a problem and clearly expressing the design process for your solution. For example, one might say "First, we thought about sending [blah blah blah], but we then realized [blah blah blah]. Then, to address that, we thought [blah blah blah], but that didn't work because of [blah blah blah]. Because of that, we ...". (Please try to avoid using the word "blah" in your answer!)

If you have any questions about what's being asked, please feel free to ask your prof.! This is a classic puzzle—I hope you have some fun with it!