Analysis of Algorithms
CS 375, Fall 2022
Problem Set 0
Due **AT THE BEGINNING OF CLASS** Wednesday, September 21

- **IMPORTANT:** Some of these exercises may build upon topics covered in our Sept. 14 class meeting; they are included here "early" so you can see all of the exercises on this assignment.
- From your textbook (CLRS), please read Chapters 1, 2.1, 2.2, and 3.
- To submit this Problem Set, the standard file naming and HW submission conventions apply: Please put your answers in a PDF file named

  `CS375_PS0_<userid>.pdf`

  where `<userid>` is replaced by your Colby userid—for example, my file would be called `CS375_PS0_eaaron.pdf`—and submit it to your `SubmittedWork` folder. Please reach out to me right away with any questions or concerns about this!
- A few essential style guidelines for writing algorithms in CS375:
  - 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/ouptuts, 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!
- *A general note for CS375:* 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 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. 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 $[2, 5, 5]$.

```
# 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, convincing explanation of the worst-case time complexity of your algorithm.

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!

2. List the following functions of n according to their order of growth—that is, how fast each function grows as n gets big—from lowest to highest:

$$(n-2)!, \quad 5\lg((n+100)^{10}), \quad 2^{2n}, \quad 0.001n^4 + 3n^3 + 1, \quad \ln^2 n, \quad \sqrt[3]{n}, \quad 3^n.$$

(As is conventional, the lg function is logarithm base 2; the ln function is the *natural logarithm*, logarithm base e; and $\ln^2 n$ is common notation for $(\ln n)^2$.) Although you don't need to explain every part of the ordering for this exercise, please give short explanations (1–2 sentences) for the following:

(a) how you know the second-smallest comes before the third-smallest; and

(b) how you know the second-largest comes after the third-largest.

NOTE: This exercise is not *directly* about the definition of big-O notation, and it does not require that definition! It's a mathematical background exercise, supporting the complexity analysis work we'll be doing as the semester goes along.

3. Prof. Snailshell of the Portland Institute of Technology (which continues to not really exist!) proposed the following algorithm for finding the distance between the two closest elements in an array of numbers.

MinDistance($A[0 \ldots n-1]$)
//Input: Array $A[0, ..., n-1]$ of numbers
//Output: Minimum distance between two elements of A
$dmin = \infty$ // min distance variable, initialized to ∞
for $i = 1$ **to** $n-1$
for $j = 0$ **to** $n-1$
if $i \neq j$ **and** $|A[i] - A[j]| < dmin$
$dmin = |A[i] - A[j]|$
return $dmin$

This is not the most time-efficient way to solve the problem! Make as many improvements as you can in Prof. Snailshell's algorithm. If you need to, you may change the algorithm altogether; if not, improve the implementation given.

(**Note:** The notation $|x - y|$ refers to the absolute value of the quantity $(x - y)$—i.e., the distance between x and y.)

4. Prof. E. Nigma of the Portland Institute of Technology hired you to analyze the algorithm given here in pseudocode, but as usual, Prof. Nigma neglected to explain what the algorithm does.

```
// Input: A matrix A[0..n-1, 0..n-1] of integers
for i = 0 to n-2 do
    for j = i+1 to n-1 do
        if A[i,j] != A[j,i]
            return False
return True
```

In the above, recall that a matrix is essentially just a two-dimensional array, so $A[i, j]$ might in some languages be written as $A[i][j]$.

(a) What does this algorithm do? Give an English description of what inputs lead to it returning True and what inputs lead to it returning False. (You do not need to give examples as part of your answer, but you are welcome to include example 2D arrays along with the English description, if it would make your answer clearer.)

(b) Using summation notation formulas (i.e., with Σ's to represent summations), give a summation expression for the number of `!=` comparisons that are made by this algorithm in the worst case (assuming its input is of the form `A[0..n-1, 0..n-1]`). Then, solve that summation—that is, find an equivalent simple formula for that expression, showing the number of `!=` operations as a function of n. (**Note:** You may want to do Smaller Assignment SA1 before doing this exercise!)

(c) Based on your answer to exercise 4b above, give the most informative *worst-case* asymptotic time complexity bound you can for this algorithm, using big-O, Θ, or Ω notation.

(d) Give the most informative *best-case* asymptotic time complexity bound you can for this algorithm. (Note that the summation you solved in exercise 4b was under *worst-case* assumptions—please think about what would make the *best-case* time complexity!)

(e) Give the most informative asymptotic bound you can on the space complexity for this algorithm, for the best case (i.e., using the least space other than that needed for the input) and the worst case (i.e., using the most space other than that needed for the input). Your explanation should include your reasoning about whether the best case and worst case for space complexity are the same or different from each other.

As usual in CS375, be sure to give concise, convincing explanations for your answers!

5. (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!