CSE 21 Summer Academy 2016
Homework 3
Due: Friday, August 5, 2016 at 11:59pm

Instructions

Homework should be done in groups of **one to two** people. You are free to change group members at any time throughout the quarter. Problems should be solved together, not divided up between partners. A **single representative** of your group should submit your work through Gradescope. Submissions must be received by 11:59pm on the due date, and there are no exceptions to this rule.

Homework solutions should be neatly written or typed and turned in through **Gradescope** by 11:59pm on the due date. No late homeworks will be accepted for any reason. You will be able to look at your scanned work before submitting it. Please ensure that your submission is legible (neatly written and not too faint) or your homework may not be graded.

Students should consult their textbook, class notes, lecture slides, instructors, TAs, and tutors when they need help with homework. Students should not look for answers to homework problems in other texts or sources, including the internet. Only post about graded homework questions on Piazza if you suspect a typo in the assignment, or if you don't understand what the question is asking you to do. Other questions are best addressed in office hours.

Your assignments in this class will be evaluated not only on the correctness of your answers, but on your ability to present your ideas clearly and logically. You should always explain how you arrived at your conclusions, using mathematically sound reasoning. Whether you use formal proof techniques or write a more informal argument for why something is true, your answers should always be well-supported. Your goal should be to convince the reader that your results and methods are sound.

For questions that require pseudocode, you can follow the same format as the textbook, or you can write pseudocode in your own style, as long as you specify what your notation means. For example, are you using "=" to mean assignment or to check equality? You are welcome to use any algorithm from class as a subroutine in your pseudocode. For example, if you want to sort list A using InsertionSort, you can call InsertionSort(A) instead of writing out the pseudocode for InsertionSort.

Required Reading Rosen Sections 3.2 and 3.3

Key Concepts Asymptotic notation, including the definitions of O, Θ, and Ω; best-case, worst-case, average-case; analyzing the run-time of algorithms

---

1. (10 points) Let $f(n)$ and $g(n)$ be functions from the nonnegative integers to the positive real numbers. Prove the following transitive property from the definition of big O:

$$\text{If } f(n) \in O(g(n)) \text{ and } g(n) \in O(h(n)) \text{ then } f(n) \in O(h(n)).$$

2. True or False? No justification needed.

   (a) (2 points) $2(3^n) \in \Theta(3(2^n))$

   (b) (2 points) $(n^6 + 2n + 1)^2 \in \Omega((3n^3 + 4n^2)^4)$

   (c) (2 points) $\log n \in \Omega(\log n + n)$

   (d) (2 points) $n \log n + n \in O(n \log n)$

   (e) (2 points) $\log(n^{10}) \in \Theta(\log(n))$

3. (10 points) In this problem, your goal is to identify who among a group of people has a certain disease. You collect a blood sample from each of the people in the group, and label them 1 through n. Suppose that you know in advance that exactly one person is infected with the disease, and you must identify who that person is by performing blood tests. In a single blood test, you can specify any subset of the samples, *combine a drop of blood from each of these samples,* and then get a result. If any sample in the subset is infected, the test will come up positive, otherwise it will come up negative. Your goal is to find the infected person with as few blood tests as possible.

   This idea of testing multiple samples at a time has been used in history at times when it was impractical or expensive to perform blood tests, for example, to find out which soldiers in World War II were carrying diseases. In those situations, the problem was even harder because there could be any number of infected people in the group. Later, we will encounter this same problem in more generality.

   Give an algorithm that finds the infected sample in a set of n blood samples, using as few tests as you can. Write pseudocode and a high-level English description of how your algorithm works. You don't need to prove the correctness of your algorithm.

4. Since some sorting algorithms are more efficient when the input list is close to sorted, we might be able to quickly sort a list by first checking how close the input list is to already being sorted, and then using that information to help us choose which sorting algorithm to use. One measurement of how close a list is to being sorted is the "move distance."

   Given an input list L of length n such that each number $1, 2, \ldots, n$ appears exactly once, define the "move distance" of L as the sum of the number of positions that each list element must be moved to get to its correct sorted position.

   Example: The move distance of $3, 4, 1, 5, 2$ is $2 + 2 + 2 + 1 + 3 = 10$ because 3 is 2 positions away from position 3, 4 is 2 positions away from position 4 and so on.

   The move distance of a list L gives us a sense of how far away list L is from being sorted.

   (a) (6 points) Write pseudocode that returns the move distance of any input list consisting of each number $1, 2, \ldots, n$ appearing exactly once.

(b) (4 points) Analyze the runtime of your algorithm using Θ notation.

5. Given two lists A of size m and B of length n, our goal is to construct a list of all elements in list A that are also in list B. Consider the following two algorithms to solve this problem.

procedure Search1(List A of size m, List B of size n)

Initialize an empty list L.
SORT* list B.
for each item $a \in A$,
  if BinarySearch$(a, B) \neq 0$ **then**
    Append a to list L.
return L

***Note:** Assume that the SORT algorithm used in Search1 takes time proportional to $k \log k$ on an input list of size k.

procedure Search2(List A of size m, List B of size n)

Initialize an empty list L.
for each item $a \in A$,
  if LinearSearch$(a, B) \neq 0$ **then**
    Append a to list L.
return L

(a) (2 points) Calculate the runtime of Search1 in Θ notation, in terms of m and n.
(b) (2 points) Calculate the runtime of Search2 in Θ notation, in terms of m and n.
(c) (2 points) When $m \in \Theta(1)$, which algorithm has faster runtime asymptotically?
(d) (2 points) When $m \in \Theta(n)$, which algorithm has faster runtime asymptotically?
(e) (2 points) Find a function $f(n)$ so that when $m \in \Theta(f(n))$, both algorithms have equal runtime asymptotically.