CSE 21 Summer Academy 2016
Homework 3
Due: Tuesday, Aug 9, 2016 at 11:59pm

Instructions

Homework should be done in groups of **one to three** 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:

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

Solution: Since $f(n)$ and $g(n)$ are always positive real numbers for all n, it is okay to ignore the absolute values in the definition of big O. If $f(n) \in O(g(n))$, this means there are constants $k \geq 0$ and $C > 0$ so that $f(n) \leq C \cdot g(n)$ for all $n > k$. Similarly, if $g(n) \in O(h(n))$, this means there are $m \geq 0$ and $B > 0$ so that $g(n) \leq B \cdot h(n)$ for all $n > m$. Then for all $n > max(k, m)$, both $f(n) \leq C \cdot g(n)$ and $g(n) \leq B \cdot h(n)$. Putting this together means $f(n) \leq CB \cdot h(n)$ for all $n > max(k, m)$, which proves $f(n) \in O(h(n))$.

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

   Solution: With explanations for your benefit - explanations were not required for this question.

   (a) (2 points) **F** **Explanation:** $\lim_{n\to\infty} \frac{2(3^n)}{3(2^n)} = \infty$

   (b) (2 points) **T** **Explanation:** Each polynomial is degree 12 and so they have the same growth rate asymptotically.

   (c) (2 points) **F** **Explanation:** It is not the case that $\log n + n \in O(\log n)$, since the largest term is $\log n + n$ is n.

   (d) (2 points) **T** **Explanation:** $n \log n$ is asymptotically greater than n so we can drop the n. Both functions are of the same order.

   (e) (2 points) **T** **Explanation:** By log rules, $\log(n^{10}) = 10 \log(n)$ and then you can drop the coefficient.

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.

Solution:

Pseudocode:
FindInfected(blood samples labeled 1 through n, exactly one of which is infected)

1. $i \leftarrow 1,\ j \leftarrow n$
2. While $i < j$
3. $\quad m \leftarrow \lfloor (i+j)/2 \rfloor$
4. $\quad$ Combine some blood from samples i through m and test.
5. $\quad$ If test is positive, set $j \leftarrow m$.
6. $\quad$ Else, set $i \leftarrow m+1$
7. Return i.

English description:

The algorithm works by repeatedly splitting the samples in two halves. It will combine drops of blood from each individual in one of the halves, and test that combination of blood. If it comes up positive, we know the infected person is among the half that we tested. If the test is negative, we know the other half contains the infected sample. This reduces our problem now to testing half as many samples as we originally had. We continue testing one half at a time until we reduce our search range down to one sample. At this point, we know the one remaining sample must be the infected sample, so we return it.

This process is essentially binary search because we are searching for our infected person by splitting the search range in half each time. Therefore, it takes about $log_2 n$ tests to narrow the search range down to a single person. We saw in class that binary search was the most efficient searching algorithm, so this solution solves the problem using as few tests as possible.

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.

Solution:

(a) **Pseudocode:**
procedure MoveDistance($a_1, a_2, \ldots, a_n$: a list of integers such that each number $1, 2, \ldots, n$ appears exactly once.)

1. $md := 0$.
2. **for** $i := 1$ to n
3. $\quad md := md + |i - a_i|$
4. **return** md.

(b) The body of the loop (line 3) takes constant time $\Theta(1)$ and this loop body occurs n times so the algorithm runtime is in $\Theta(n)$.

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)

1. Initialize an empty list L.
2. SORT* list B.
3. **for** each item $a \in A$,
4. $\quad$ **if** BinarySearch$(a, B) \neq 0$ **then**
5. $\quad\quad$ Append a to list L.
6. **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)

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

(a) (2 points) Calculate the runtime of Search1 in Θ notation, in terms of m and n.

Solution: $\Theta(n \log n + m \log n)$. In Search1, line 1 is constant time, then in line 2, we sort list B, which takes times $n \log n$ since list B is a list of size n. Then, for each element of A, we do a binary search in a list of size n. Each such binary search takes times $\log n$ and we do m such searches, so the time of lines 3 through 5 is $m \log n$. Line 6 is also constant time. Since the time of consecutive pieces of code comes from their sum, we know the whole algorithm takes time $\Theta(n \log n + m \log n)$. Note that we can't drop either term because we don't know which is larger, m or n.

(b) (2 points) Calculate the runtime of Search2 in Θ notation, in terms of m and n.

Solution: $\Theta(mn)$. In Search2, line 1 is constant time, then we do a linear search in a list of size n a total of m times. Since linear search takes time proportional to n, and we do this in a loop that runs m times, the total runtime is $\Theta(mn)$.

(c) (2 points) When $m \in \Theta(1)$, which algorithm has faster runtime asymptotically?

Solution: Search2. When $m \in \Theta(1)$, it means m is a constant. Letting m be a constant in our answers for parts (a) and (b), we see Search 1 has runtime $\Theta(n \log n)$ and Search 2 has runtime $\Theta(n)$, so Search 2 is faster.

(d) (2 points) When $m \in \Theta(n)$, which algorithm has faster runtime asymptotically?

Solution: Search1. When $m \in \Theta(1)$, it means m is a linear function in n, say cn where c is a constant. If we replace m with cn in our answers for parts (a) and (b), we see Search 1 has runtime $\Theta(n \log n)$ and Search 2 has runtime $\Theta(n^2)$, so Search 1 is faster.

(e) (2 points) Find a function $f(n)$ so that when $m \in \Theta(f(n))$, both algorithms have equal runtime asymptotically.

Solution: $f(n) = \log n$. If we let $m \in \Theta(\log n)$, it means m is proportional to the log of n, say $m = c \log n$ where c is a constant. Replacing m with $c \log n$ in our answers for parts (a) and (b) shows that both Search1 and Search2 will have a runtime of $\Theta(n \log n)$.