Algorithms
Computer Science 140 & Mathematics 168
Spring 2012
Homework 2b
Due Tuesday, January 31

- This assignment has a bonus problem! The bonus problems are optional but are highly recommended. (They are also very similar in spirit to technical interview questions that many companies use when hiring!)
- Please remember to use LaTeX and submit each problem separately.
- Be sure to use big-Θ rather than big-Oh for asymptotic running time when you can. Remember, big-Θ is stronger than big-Oh. Similarly, always use big-Ω for lower-bounds.

1. **[20 Points] Professor Mae Tresees!** Professor Mae Tresees has devised an algorithm for computing the Treseesian function on two $n \times n$ matrices. (Nevermind what that function does - this is the gratuitous story!).

   (a) Professor Tresees' first algorithm has a worst-case running time described by the recurrence relation:

   $$\begin{aligned} T(1) &= c \\ T(n) &= 8T(n/2) + cn^4 \end{aligned}$$

   What is the asymptotic running time of this algorithm as a function of n? Show your work!

   (b) By using some very clever algebra, Professor Tresees has removed one of the recursive calls and now has an algorithm with a worst-case running time described by the recurrence relation:

   $$\begin{aligned} T(1) &= d \\ T(n) &= 7T(n/2) + dn^4 \end{aligned}$$

   Professor Tresees is excited because she remembers that Strassen's matrix multiplication algorithm was so fast because it reduced the number of recursive calls from 8 to 7. Now, Professor Tresees wants to see if that will help here. What is the asymptotic running time of this algorithm as a function of n? Show your work!

(c) Is the second algorithm asymptotically better than the first in this case? Briefly, what is the reason for this outcome?

2. **[20 Points] Sorting with Professor I. Lai!** Professor I. Lai of the Pasadena Institute of Technology (P.I.T.) has hired you as a research assistant.

   (a) "Here's a recurrrence relation that I have been studying where $\ell > 1$ is some positive integer and c is some constant," says Prof. Lai.

   $$T(n) = \ell \times T(n/\ell) + cn$$

   $$T(1) = c$$

   Use the work tree method to solve this recurrence relation. Show your work. Leave your answer as a function of n, ℓ, and c (and no other variables). **If you have any logarithms in that expression, leave them in base ℓ.**

   (b) "Now let me tell you about a new sorting algorithm that I've developed. It's just like Mergesort except that rather than dividing the array into two (more-or-less) equal sized halves, it divides the array into $\ell > 1$ pieces of size (more-or-less) n/ℓ. It then sorts each of those pieces recursively and merges those ℓ sorted lists into one sorted list. You've solved the recurrence relation for me - thank you! Notice that when $\ell = 2$ your solution is (or should be!) $O(n \log n)$, which is what I expect because that's the known running time of Mergesort. However, I'm going to choose ℓ to be equal to n. What running time do we get now?" Plug in $\ell = n$ into your solution for the recurrence relation to get the running time of the sorting algorithm and show the answer.

   (c) "This is beyond fantastic!" exclaims Prof. Lai. What about this **running time** that you got for Lai's sorting algorithm (when $\ell = n$) tells you that *something* is definitely wrong?

   (d) In fact, there's nothing wrong with choosing $\ell = n$. What's wrong is that the recurrence relation above is not quite the correct recurrence relation for the algorithm that Prof. Lai has described. Write down the corrected recurrence relation and explain briefly why this new recurrence relation is correct for Lai's algorithm.

   (e) Solve the new recurrence relation with the work-tree method. Show your work. Again, leave your answer as a function of n, ℓ, and c (and no other variables).

(f) Now, what is the running time of Prof. Lai's algorithm when $\ell = 2$, $\ell = 42$, and when $\ell = n$? What values of ℓ give asymptotically "good" running times and which give asymptotically "bad" running times?

3. **[20 Points] The Index Problem!** Let A be an array of n distinct integers where A is already **sorted** in ascending order. Our problem is to find an index $i, 1 \leq i \leq n$, such that $A[i] = i$ or determine that no such i exists.

   (a) Describe an algorithm for this problem with $O(\log n)$ running time. You should give the algorithm (in clear English or in clear high-level pseudo-code) and briefly explain why the running is $O(\log n)$ in the worst case.

   (b) Show that any comparison-based algorithm for this problem must have $\Omega(\log n)$ worst-case running time. (*Note:* Use an argument very similar to the $\Omega(n \log n)$ lower bound we achieved for comparison-based sorting.)

   (c) Aha! So your algorithm from part (a) runs in $O(\log n)$ time and $\log n$ is an asymptotic lower bound for solving this problem – this implies that you've found an asymptotically optimal algorithm for this problem! Eat chocolate to celebrate! (This part of the problem is worth ϵ points.)

4. **[20 Points] Sorting Partially Sorted Data.** You are given an array of n elements to sort. The good news is that the array is already partitioned into n/k blocks of k elements each. The elements in the first block (elements at array indices 1 through k) are unsorted, but they are *all* less than the elements in the second block (elements at array indices $k+1$ through $2k$), and so forth. In other words, each of the n/k blocks is unsorted, but the elements in each block are strictly smaller than the elements in the next block.

   Prove that any comparison-based sorting algorithm that receives this kind of "partially" sorted data has a lower bound of $\Omega(n \log k)$ on its worst-case running time.

   Note: It is not at all rigorous nor correct to simply combine the $\Omega(k \log k)$ lower bounds for sorting each of the n/k blocks! To see why, observe that a skeptic could rightfully ask if there might not be a special clever algorithm that exploits the information about the blocks to do better than we would without knowing that information. A rigorous proof will need to follow the paradigm that we used in class to get the lower bound for sorting in general.

5. **[20 Point OPTIONAL BONUS PROBLEM] The FletNix Inversions Problem!**

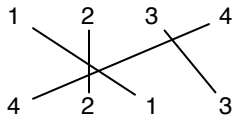

Figure 1: We count the number of inversions between two sequences by drawing lines between the corresponding numbers, no three lines intersecting at the same point, and count the number of intersections.

FletNix is an exciting new website that offers DVD rentals by mail. One of the features of the FletNix website is that subscribers are periodically asked to rank a list of n popular movies in order of preference. For example, if $n = 3$ and the movies are "named" 1 through 3, then the ranking $3, 1, 2$ says that movie 3 was your favorite, movie 1 was your second favorite, etc.

Henceforth in this problem, a ranking simply refers to a permutation of the integers 1 through n.

FletNix would like to take pairs of subscribers and measure the similarity of their rankings. Subscribers with similar rankings are then placed into groups so that FletNix can say to you "Subscribers like you have been renting Terminator 7."

How do we measure the similarity of two rankings? One way is to count the number of *inversions* between the two rankings. An inversion occurs whenever one ranking prefers i over j while the other prefers j over i. An easy way to visualize the number of inversions between rankings is to line the two rankings up, one above the other, and draw straight lines between the corresponding numbers. The lines are drawn so that no three lines intersect at the same point. For example, in the figure below, we use this method to find that the number of inversions between the ranking $1, 2, 3, 4$ and $4, 2, 1, 3$ is 4. We would say that the difference between these two rankings is 4. Notice that two identical rankings have a difference of 0.

(a) Show that two rankings of the numbers 1 through n can have $\Omega(n^2)$ inversions. That is, the difference between two rankings can grow as n^2.

(b) Describe a simple algorithm for computing the number of inversions between two rankings. What is the running time of your algorithm?

(c) Describe an algorithm that computes the number of inversions between two rankings in worst-case time asymptotically *faster* than n^2. (*Hint: It's*

possible to do this in time $O(n \log n)$.) For simplicity, assume that one of the two rankings is simply the sequence $1, 2, \ldots, n$ and the other ranking is an arbitrary permutation of the numbers 1 through n.

(d) Explain briefly but convincingly why your algorithm is correct. You don't need to give a formal proof here, but your reasoning should be sufficiently clear that it could be converted into a rigorous proof without much effort.

(e) Explain how you derived the worst-case running time for your algorithm.

(f) Explain how your algorithm can be modified to handle the case of two arbitrary permutations of the numbers 1 through n without increasing the asymptotic running time.