Algorithms
Computer Science 140 & Mathematics 168
Spring 2009
Homework 2b
Due Tuesday, February 3

Please use LaTeX to typeset your solutions to this assignment. (This is the last assignment for which LaTeXwill be required, although you are certainly encouraged to continue using it.)

1. [**15 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 big-Θ 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:

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

   What is the big-Θ 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?

2. [**15 Points**] **Order Statistics Revisited.** In class we showed that the recursive **Select** algorithm runs in time $\Theta(n \lg n)$ if the array is partitioned into groups of 3 but runs in time $\Theta(n)$ if the array is partitioned into groups of 5. Now we'll investigate what happens if the

algorithm is implemented so that the array is partitioned into groups of 7. Give the recurrence relation that arises when groups of 7 are used, and explain why your recurrence relation is correct. Then use the analysis technique that we used in class to derive the Big-Theta running time of this variant of the algorithm. Explain every step of your analysis very carefully! Hmmm, very interesting!

3. [$15+\epsilon$ **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 $i, 1 \le i \le n$, 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 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] The Pipeline Problem!** The Republic of Shmorbodia is well known for its huge oil deposits. You've been hired by the Shmorbodian People's Agency of Minerals (SPAM) to help them decide where to place a new oil pipeline.

   The pipeline must run perfectly east-west (that is, it is a horizontal line). You are given a large number, n, of (x, y) coordinate pairs where each pair defines the location of an oil well. An oil well will be connected to the pipeline by a vertical segment from the (x, y) location of the oil well to the oil pipeline. For example, the Figure 1 shows one possible scenario where the wells are represented by circles and the pipeline is represented by the thick horizontal line.

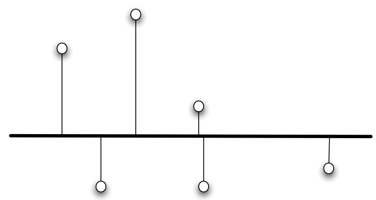

Figure 1: Wells (circles) connected to a horizontal pipeline.

Given an unordered list of n (x, y) pairs, your objective is to find the y-coordinate for the oil pipeline that minimizes the total length of all of the vertical segments used to connect the wells to the pipeline.

Describe a $O(n)$ algorithm for this problem and explain clearly and convincingly why the algorithm is correct. (If you use any algorithm that we have described in class as part of your algorithm, you may assume that the algorithm from class is correct.)

5. [**15 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.

6. [**20 Point OPTIONAL BONUS PROBLEM**] **The FletNix Inversions Problem!**

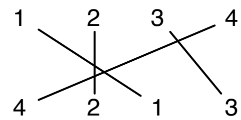

Figure 2: We count the number of inversions between two sequences by drawing lines between the corresponding numbers, no three lines intersecting at the same point, and count the number of intersections.

Please remember to submit bonus problems on a separate sheet from your regular homework submission.

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.