CS 70 Discrete Mathematics for CS Fall 2003 Wagner — HW 3

Due Thursday September 18

Formatting your solution: Please put at the top of your solution the following information: your username on `cory.eecs`, your full name, the string "CS70, Fall 2003, HW 3", your section number, and your partners.

1. **(5 pts.) Any questions?**
   What's the one thing you'd most like to see explained better in lecture or discussion sections?

2. **(10 pts.) Making big rocks into little rocks**
   In the mound-splitting game, one starts off with a single mound of pebbles. In each move, one picks a mound, splits it into two mounds of arbitrary size (say, k and m pebbles), multiplies the number of pebbles in each of the two new mounds, and writes down the result (i.e., $k \times m$). One keeps playing until there is no mound with more than one pebble, and then one adds up all of the numbers previously written down. Prove that if one starts with a mound of n pebbles, the final sum will be $n(n-1)/2$, *no matter what order the moves are done in*.

3. **(10+10 pts.) Share the wealth**
   A wealthy king arranges n^2 gold pieces in a $n \times n$ grid, each with their "heads" side up. He invites all comers to play a game: in each move, you can pick one of the n rows or columns and flip over all of the gold pieces in that row or column, but you are not allowed to re-arrange them in any other way. The king announces that anyone who is able to reach a configuration where there is exactly one gold coin with its "tails" side up will win as a prize all n^2 of the gold pieces. The king generously allows each participant an unlimited number of moves. For which values of n is it possible to win the game? Prove your answer.

   Optional bonus problem: Suppose the king relaxes the rules, so that you win if the number of "tails" is between 1 and $n-1$. Now which values of n allow one to win? Prove your answer.

4. **(10 pts.) Mergesort**

   (a) Write a complete recursive definition of the algorithm Merge, which takes two sorted lists u and v as arguments and returns a sorted list that is the result of merging u and v. [Feel free to use pseudo-code.]

   (b) Prove by induction that your implementation terminates and that no more than $n-1$ comparisons are required to merge two disjoint lists of combined length n.

5. (65 pts.) The Closest Pair Problem

You are given the positions of n points in the plane. How would you determine a pair of distinct points which are closest together? This is an example of the kind of problem that occurs in such application areas as air traffic control, the analysis of galaxies in astronomy, and the layout of integrated circuits. In this problem, we will develop a divide-and-conquer algorithm that solves the problem much more efficiently than naïve methods.

(a) Consider the naïve algorithm that simply computes the distance between every pair of distinct points and keeps track of the closest pair found. Show that the running time of this algorithm is $O(n^2)$.

(b) Figure 1 shows a template for a divide-and-conquer algorithm for this problem. (We assume as usual for simplicity that n is a power of two. Also for simplicity, the algorithm only outputs the minimum distance found; it will be clear shortly how to modify it so that it outputs a closest pair of points.)

Prove by induction on n that the algorithm Closest_Pair is correct, i.e., for any input set P of n points ($n \geq 2$ a power of two), the algorithm outputs the minimum distance between any pair of points in P.

(c) Let's now consider the implementation of step $(*)$. Denote by S the subset of P consisting of all points in a vertical strip of width $2d$ centered on the dividing line Λ. Show that in this step, it is enough to consider only pairs of points p, q where $p \in P_L \cap S$ and $q \in P_R \cap S$.

(d) Prove that in any square box of side $d/2$ there can be at most one point from P_L and at most one point from P_R.

(e) Now use parts (c) and (d) to prove that, for each point $p \in P_L \cap S$, there can be at most 8 points q in $P_R \cap S$ whose distance from p is less than d. [The number 8 may be an over-estimate: if you can do better, that is good but not important for our purposes.]

(f) From part (e), it should be reasonably clear that the $(*)$ step requires at most $8n$ distance computations. [Make sure you understand why.] Use this fact to prove by induction on n that the maximum number, $D(n)$, of distance computations performed by the algorithm Closest_Pair on an input of n points (n a power of two, $n \geq 2$) satisfies $D(n) \leq 8n \lg n - 15n/2$.

Algorithm Closest_Pair(P)
$\{P$ *is a set of n points in the plane,* $n \geq 2$ *a power of two;*
the points are assumed to be sorted by x-coordinate and by y-coordinate$\}$
if $|P| = 2$ then output the distance d between the two points in P
else
divide P by x-coordinate into sets P_L, P_R, each of size $\frac{n}{2}$, using a vertical line Λ
$d_L :=$ Closest_Pair(P_L)
$d_R :=$ Closest_Pair(P_R)
$d := \min\{d_L, d_R\}$
$(*)$ check whether there exists a pair of points p, q with $p \in P_L$ and $q \in P_R$
whose distance is less than d; let d' be the minimum such distance found,
and $d' = +\infty$ if no such pair exists
output $\min\{d, d'\}$

Figure 1: Template for a closest-pair algorithm. The implementation of step $(*)$ is left deliberately vague; you will fill in the details of this in parts (c)–(g). In the "divide" step, it is possible that one or more points may lie on the vertical dividing line Λ. In this case we are free to allocate these points arbitrarily to P_L or P_R, as long as we ensure an even split. "Sorted by x-coordinate and by y-coordinate" means that there are two sorted lists containing the same set of points, one in x-order and one in y-order. You need not worry for now about how these two lists are constructed for P_L and P_R.