CS 473: Fundamental Algorithms, Spring 2013

Discussion 4

February 6, 2013

4.1. Recurrences

Solve the following recurrences.

(A) $T(n) = 5T(n/4) + n$ and $T(n) = 1$ for $1 \leq n < 4$.

(B) $T(n) = 2T(n/2) + n \log n$

(C) $T(n) = 2T(n/2) + 3T(n/3) + n^2$

4.2. Tree Traversal.

Let T be a rooted binary tree on n nodes. The nodes have unique labels from 1 to n.

(A) Given the preorder and postorder node sequences for T, give a recursive algorithm to reconstruct a tree that satisfies the preorder and postorder sequences. Is this reconstruction unique?

(B) Given the preorder and inorder node sequences for T, give a recursive algorithm to reconstruct a tree that satisfies the preorder and inorder sequences. Is this reconstruction unique?

4.3. Divide and Conquer.

Let $p = (x, y)$ and $p' = (x', y')$ be two points in the Euclidean plane given by their coordinates. We say that p dominates p' if and only if $x > x'$ *and* $y > y'$. Given a set of n points $P = \{p_1, \ldots, p_n\}$, a point $p_i \in P$ is undominated in P if there is no other point $p_j \in P$ such that p_j dominates p_i. Describe an algorithm that given P outputs all the undominated points in P; see figure. Your algorithm should run in time asymptotically faster than $O(n^2)$.

4.4. Merging arrays.

Suppose you are given k sorted arrays $A_1, A_2, \ldots, A_k$ where each array contains n elements. The goal is to merge all the arrays into a single sorted array A of kn elements. Given two

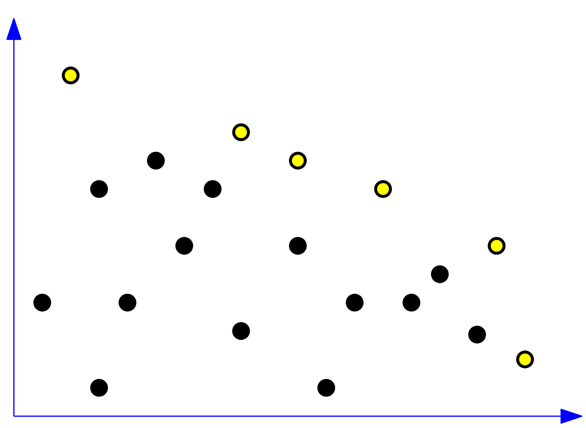

Figure 1: The undominated points are shown as unfilled circles.

sorted arrays of size x and y respectively, you know that they can be merged into a single sorted array in $O(x + y)$ time.

(A) Suppose you use the following algorithm for merging the k arrays. Merge A_1 and A_2. Merge the resulting array with A_3 and the result with A_4 and so on. What is the running time of this algorithm as a function of k and n?

(B) Give a more efficient algorithm using divide and conquer.

(C) Consider the following modification to the merge sort algorithm. Instead of splitting the input array into 2 subarrays, recursively sorting each and merging the 2 sorted subarrays, we will split the input array into k subarrays, recursively sort each (using the modified algorithm), and merge the k sorted subarrays. How does the running time of the modified algorithm compare to that of the original algorithm?

4.5. Convex hull

You are given a set P of n points in the plane, and you would like to compute their convex-hull (i.e., that is the shortest perimeter polygon that contains all the points). To see how the convex-hull looks like, think about the plane as being a wood board, and place a nail at each point. Now, you shrink a rubber band around the points. The rubber shrinks into the convex-hull. Clearly, the vertices of the convex-hull are a subset of the input points. Show an $O(n \log n)$ time algorithm for computing the convex-hull. (Hint: Split the plane by a vertical line, compute the convex-hulls on both sides, and then figure out how to stitch the two convex-hulls together. To get a handle on this stitching problem, find closest points in the x-axis between the two hulls, and climb up to the stitching bridges.)