CS303 (Spring 2008)- Solutions to Assignment 8

Exercise 9.3-5

There are two ways. One is to use it as a subroutine in deriving a new Divide&Conquer algorithm, and reanalyze it. That's not terribly difficult. Even easier is to reduce it as follows:

Suppose you wan to find the k^{th} smallest element in an array `a` of n elements. First, with one linear scan in time $\Theta(n)$, we find the minimum min and maximum max of the array. Now, if $k \leq n/2$, we add $n-2k$ copies of $\min -1$ to the array. Otherwise, we add $2k-n$ copies of $\max +1$ to the array. The new number of elements in these two cases is $2(n-k)$ or $2k$, respectively. The previously k^{th} element will now be at position $k+(n-2k) = n-k$ in the former case, and still in position k in the latter case. Thus, what was previously the k^{th} element is the median of the new array. Thus, after this modification (which took linear time), we can simply run the linear-time subroutine for finding the median, and it will return the right element.

Exercise 9.3-8

The idea is the following. Compare elements $X[n/2]$ and $Y[n/2]$. If $X[n/2] = Y[n/2]$, then either of them is the median overall, since half of the elements of both X and Y are smaller, and half are larger. Otherwise, assume that $X[n/2] > Y[n/2]$ (if $X[n/2] < Y[n/]$, rename the arrays so this holds).

If $X[n/2] \leq Y[n/2]+1$, then $X[n/2]$ is a median (depending on the tie breaking). If not, then all the elements $X[n/2]+1, \ldots, X[n]$ are larger than all of $X[1], \ldots, X[n/2]$ as well as $Y[1], \ldots, Y[n/2+1]$, and therefore cannot be the median.

Similarly, if $X[n/2-1] \leq Y[n/2]$, then $Y[n/2]$ is a median (again depending on tie breaking). If not then all the elements $Y[1], \ldots, Y[n/2-1]$ are smaller than all of $Y[n/2], \ldots, Y[n]$ and all of $X[n/2-1], \ldots, X[n]$, and therefore cannot be medians.

Thus, unless we have found a median (which took us constant time to test), we can throw out all of $Y[1], \ldots, Y[n/2-1]$ and all of $X[n/2+1], \ldots, X[n]$, leaving us with two arrays of size $n/2+1$ each. Once we reach size 1, either number is a median. This gives us the running time recurrence $T(n) = T(n/2)+c$, with solution $T(n) = \Theta(\log n)$.

Problem 9-2

(a) If all the weights are $1/n$, then the constraint $\sum_{x_i<x_k} w_i < 1/2$ is saying that $\sum_{x_i<x_k} 1/n < 1/2$, i.e., $\sum_{x_i<x_k} 1 < n/2$. This is equivalent to saying that the number of i with $x_i < x_k$ is $n/2$. Similarly, the other constraint $\sum_{x_i>x_k} w_i \leq 1/2$ says that the number of elements i with $x_i > x_k$ is at most $n/2$. But having at most half the elements smaller, and half larger is exactly the definition of a median.

(b) First sort the elements in time $O(n \log n)$. Now start from the left, and sum up weights until the sum of weights is about to cross $1/2$ once one more element is added in. The next element k about to be added is the weighted median. (The total weight of elements larger than k is also at most $1/2$; otherwise, element k could have been safely added.) The second stage only takes $O(n)$, so the overall time is $O(n \log n)$.

(c) The algorithm is as follows. First, we ignore the weights, and compute the regular median x_m of the n elements in time $O(n)$ using the algorithm from class. Now, in one linear pass through the array, we compare all the elements x_i to the median, and sum up the weights w_i of all elements x_i with $x_i < x_m$, and separately all the elements x_j with $x_j > x_m$, and also all x_i with $x_i = x_m$. Call those weight sums W^-, W^+, and $W^=$. If both W^-, W^+ are at most $1/2$, then m works as a weighted median.

Otherwise, exactly one of W^-, W^+ is larger than $1/2$ (they can't both be, because the weights sum to 1). If the sum of weights of $x_i < x_m$ is too large, then we know that the correct solution must be one of those x_i. Otherwise, the solution must be one of the x_j with $x_j < x_m$. In either case, we recurse

on the corresponding set of remaining items, of size at most $n/2$. (We can safely throw out all x_i with $x_i = x_m$.) The only slight wrinkle is that we also need to remember what weight we are aiming for, e.g., if we recurse on the right side, we don't want the weighted *median* of that set, but need to take into consideration the elements already known to be smaller.

Thus, we introduce a parameter ω, the weight we are aiming for. If W^- was too large, we simply recurse with the same ω. Otherwise, we recurse with $\omega - W^- - W^=$ on the larger elements.

The running time will be $T(n) = T(n/2) + O(n)$, which by Master Theorem has solution $T(n) = \Theta(n)$.

(d) Let x be the location of the weighted median, and consider some location $y = x - \delta$ (to the left of x). For each point to the left of y, the distance to the post office decreases by δ. For each point to the right of x, it increases by δ. For points between y and x, it can certainly not do better than decrease by δ. Thus, the total weight of points for which the distance decreases is at most $1/2$, and the total weight of points for which it increases is at least $1/2$. Thus, the total weighted distance increases by going from x to $x - \delta$. The same argument works for going from x to $x + \delta$.

(e) For the 2-dimensional version, we simply compute the weighted median of the x and y coordinates separately. Call them m_x and m_y. We then output the point (m_x, m_y). The nice thing about the Manhattan distance is that the (weighted) sum of Manhattan distances is the weighted sum of the first coordinate distances, plus the weighted sum of the second coordinate distances. (This does not hold for Euclidean distances.) Thus, minimizing each separately minimizes the total.

Formally, the weighted sum of distances is

$$\sum_i w_i d(p, p_i) \quad = \quad \sum_i w_i (|x_i - x_p| + |y_i - y_p|) \quad = \quad \sum_i w_i |x_i - x_p| + \sum_i w_i |y_i - y_p|.$$

Since we can choose any coordinates, we should choose x_p, y_p such that the individual sums are minimized, which is exactly the proposed solution.

Exercise 15.3-3

Yes, this problem also exhibits optimal substructure. If we know that we need the subproduct $(A_\ell \cdot A_r)$, then we should still find the most expensive way to compute it — otherwise, we could do better by substituting in the most expensive way.

Exercise 15.3-5

Here is a simple example. The matrix sizes are, in order, $6 \times 7, 7 \times 3, 3 \times 1$. If we multiply the first two matrices first, the cost is $6 \cdot 7 \cdot 3 = 126$, and the second multiplication will have cost $6 \cdot 3 \cdot 1 = 18$, for a total of 144. The other order will first have cost $7 \cdot 3 \cdot 1 = 21$, and the second multiplication will have cost $6 \cdot 7 \cdot 1 = 42$, for a total of 63. So clearly, the second is better. However, the chosen split would be $k = 2$, since $6 \cdot 3 \cdot 1 < 6 \cdot 7 \cdot 1$.