CMPS 102 Solutions to Homework 4

Solutions by Cormen and us

October 20, 2005

Problem 1. 8.2-4 p.170
Compute the C array as in done in counting sort. The number of integers in the range $[a \dots b]$ is $C[b] - C[a-1]$, where we interpret $C[-1]$ as 0.

Problem 2. 8.3-2 p.173
Insertion sort is stable. When inserting $A[j]$ into the sorted sequence $A[1 \dots j-1]$, we do it the following way: compare $A[j]$ to $A[i]$, starting with $i = j-1$ and going down to $i = 1$. Continue as long as $A[j] < A[i]$.
Merge sort as defined is stable, because when two elements compared are equal, the tie is broken by taking the element from array L which keeps them in the original order.
Heapsort and quicksort are not stable.
One scheme that makes a sorting algorithm stable is to store the index of each element (the element's place in the original ordering) with the element. When comparing two elements, compare them by their values and break ties by their indices.
Additional space requirements: For n elements, their indices are $1 \dots n$. Each can be written in $\lg n$ bits, so together they take $O(n \lg n)$ additional space.
Additional time requirements: The worst case is when all elements are equal. The asymptotic time does not change because we add a constant amount of work to each comparison.

Problem 3. 8.3-4 p.173
Treat the numbers as 2-digit numbers in radix n. Each digit ranges from 0 to $n-1$. Sort these 2-digit numbers with radix sort.
There are 2 calls to counting sort, each taking $\Theta(n+n) = \Theta(n)$ time, so that the total time is $\Theta(n)$.

Problem 4. 8.4-2 p.177
The worst-case running time for the bucket-sort algorithm occurs when the assumption of uniformly distributed input does not hold. If, for example, all the input ends up in the first bucket, then in the insertion sort phase it needs to sort all teh input, which takes $O(n^2)$ time.
A simple change that will preserve the linear expected running time and make the worst-case running time $O(n \lg n)$ is to use a worst-case $O(n \lg n)$-time al-

gorithm like merge sort instead of insertion sort when sorting the buckets.

Problem 5. 9.3-3 p.192
A modification to quicksort that allows it to run in $O(n \lg n)$ time in the worst case uses the deterministic PARTITION algorithm that was modified to take an element to partition around as an input parameter.
SELECT takes an array A, the bounds p and r of the subarray in A, and the rank i of an order statistic, and in time linear in the size of the subarray $A[p \ldots r]$ it returns the ith smallest element in $A[p \ldots r]$.

```
BEST-CASE-QUICKSORT($A, p, r$)
1. if $p < r$
2.     then $i \leftarrow \lfloor (r - p + 1)/2 \rfloor$
3.         $x \leftarrow$ SELECT($A, p, r, i$)
4.         $q \leftarrow$ PARTITION($x$)
5.         BEST-CASE-QUICKSORT($A, p, q - 1$)
6.         BEST-CASE-QUICKSORT($A, q + 1, r$)
```

For an n-element array, the largest subarray that BEST-CASE-QUICKSORT recurses on has $n/2$ elements. This situation occurs when $n = r - p + 1$ is even; then the subarray $A[q+1 \ldots r]$ has $n/2$ elements, and the subarray $A[p \ldots q-1]$ has $n/2 - 1$ elements.
Because BEST-CASE-QUICKSORT always recurses on the subarrays that are at most half the size of the original array, the recurrence for the worst-case running time is $T(n) \leq 2T(n/2) + \Theta(n) = O(n \lg n)$.

Problem 6. 9.3-8 p.193

Solution 1: As presented in lecture.
We are given X and Y, two sorted arrays of length n. The median of the combined array of X and Y is less than or equal to n elements and greater than or equal to $n - 1$. We find the medians of X and Y, m_x and m_y respectively. Compare m_x and m_y. The base case is when $n = 1$, then the minimum of $\{m_x, m_y\}$ is the median. Otherwise, $m_x \leq m_y$ or $m_y \leq m_x$. Without loss of generality, suppose that $m_x \leq m_y$, then there are $\lfloor \frac{n}{2} \rfloor$ elements of X that are equal or less than m_x and cannot be the median, since there are less than $n - 1$ elements of X and Y that are less than or equal to m_x. Similarly, there are $\lceil \frac{n}{2} \rceil$ elements of Y that are greater than or equal to m_y and cannot be the median. However, we need to keep the size of the two arrays we recurse on equal, so we eliminate the $\lfloor \frac{n}{2} \rfloor$ greatest elements of Y and the $\lfloor \frac{n}{2} \rfloor$ smallest elements of X. After eliminating $\lfloor \frac{n}{2} \rfloor$ elements from each list, we recurse on the smaller lists by finding and comparing the new medians, and eliminating half the elements.

Example: We consider a simple example where $X = \{1, 3, 4, 6, 10\}$ and $Y = \{2, 5, 7, 8, 9\}$. The median of X and Y is 5 ($\{1, 2, 3, 4\} \leq 5 \leq \{6, 7, 8, 9, 10\}$). Fist we find the medians of X and Y, $m_x = 4 < 7 = m_y$. There are only three elements less than $m_x = 4$, so it cannot be the median. We eliminate the elements less than m_x in X; the resulting array is $X' = \{4, 6, 10\}$. By a similar argument, $m_y = 7$ cannot be the median, so we eliminate the elements in Y that are greater than m_y, and the resulting array is $Y' = \{2, 5, 7\}$. Now we find the medians of X' and Y', $m'_x = 6 > 5 = m'_y$. We eliminate 10 from X' to get $X'' = \{4, 6\}$, and eliminate 2 from Y' to get $Y'' = \{5, 7\}$. Once more, we find $m''_x = 4 < 5 = m''_y$, and get $X''' = \{6\}$ and $Y''' = \{5\}$. We have reached the base case of the algorithm, so we compare $m'''_x = 6 > 5 = m'''_y$, and return 5.

Analysis: At each stage of the recursion we do a constant number of comparisions, and we remove $\lfloor \frac{n}{2} \rfloor$ elements from each list. So the recurrence equation is $T(n) = T(\lceil \frac{n}{2} \rceil) + \Theta(n)$, and $T(n) \in \Theta(\lg n)$.

Solution 2: A binary search like algorithm
Let's start out by supposing that the median (the lower median, since we know we have an even number of elements) is in X. Let's call the median value m, and let's suppose that it's $X[k]$. Then k elements of X are less than or equal to m and $n - k$ elements of X are greater than or equal to m. We know that in the two arrays combined, there must be n elements less than or equal to m and n elements greater than or equal to m, and so there must be $n - k$ elements of Y that are less than or equal to m and $n - (n - k) = k$ elements of Y that are greater than or equal to m.
Thus, we can check that $X[k]$ is the lower median by checking whether $Y[n-k] \leq X[k] \leq Y[n - k + 1]$. A boundary case occurs for $k = n$. Then $n - k = 0$, and there is no array entry $Y[0]$; we only need to check that $X[n] \leq Y[1]$.
Now, if the median is in X but is not $X[k]$, then the above condition will not hold. If the median is $X[k']$, where $k' < k$, then $X[k]$ is above the median, and $Y[n - k + 1] < X[k]$. Conversely, if the median is $X[k'']$, where $k'' > k$, then $X[k]$ is below the median, and $X[k] < Y[n - k]$.
Thus we can use a binary search to determine whether there is an $X[k]$ such that either $k < n$ and $Y[n-k] \leq X[k] \leq Y[n-k+1]$ or $k = n$ and $X[k] \leq Y[n-k+1]$; if we find such an $X[k]$, then it is the median. Otherwise, we know that the median is in Y, and we use a binary search to find a $Y[k]$ such that either $k < n$ and $X[n - k] \leq Y[k] \leq X[n - k + 1]$ or $k = n$ and $Y[k] \leq X[n - k + 1]$; such a $Y[k]$ is the median. Since each binary search takes $O(\lg n)$ time, we spend a total of $O(\lg n)$ time.

We illustrate two cases:
Case 1: median $= X[k]$, where $k = \lfloor (low + high)/2 \rfloor$

$$X[1], X[2], \ldots, X[low], \ldots, X[k], \ldots, X[high], \ldots, X[n-1], X[n]$$

Then k elements are less than or equal to $X[k]$. So $n - k$ elements of Y are less than or equal to $X[k]$, and we determine that $Y[n-k] \leq X[k] \leq Y[n-k+1]$.

Case 2: median $\neq X[k]$. Without loss of generality, suppose that the median is greater than $X[k]$. There are k elements of X less than or equal to $X[k]$, but $X[k] < Y[n-k]$. This means that there are less than $n-k$ elements of Y less than or equal to $X[k]$. So the median cannot be $X[k]$.

Here's how to write the algorithm in pseudocode:

```
TWO-ARRAY-MEDIAN($X, Y$)
1. $n \leftarrow length[X]$
2. $median \leftarrow$ FIND-MEDIAN($X, Y, n, 1, n$)
3. if $median$ = NOT-FOUND
4.     then $median \leftarrow$ FIND-MEDIAN($Y, X, n, 1, n$)
5. return $median$
```

```
FIND-MEDIAN($A, B, n, low, high$)
1. if $low > high$
2.     then return NOT-FOUND
3.     else $k \leftarrow \lfloor (high + low)/2 \rfloor$
4.         if $k = n$ and $A[n] \leq B[1]$
5.             then return $A[n]$
6.         elseif $k < n$ and $B[n-k] \leq A[k] \leq B[n-k+1]$
7.             then return $A[k]$
8.         elseif  $A[k] > B[n-k+1]$
9.             then return FIND-MEDIAN($A, B, n, low, k-1$)
10.          else return FIND-MEDIAN($A, B, n, k+1, high$)
```

Extra Credit. 9-2 p.194
a. The median x of the elements $x_1, x_2, \ldots, x_n$ is an element $x = x_k$ satisfying $|\{x_i : 1 \leq i \leq n \text{and} x_i < x\}| \leq n/2$ and $|\{x_i : 1 \leq i \leq n \text{and} x_i > x\}| \leq n/2$. If each element x_i is assigned a weight $w_i = 1/n$, then we get

$$\begin{aligned}
\sum_{x_i < x} w_i &= \sum_{x_i < x} \frac{1}{n} \\
&= \frac{1}{n} \cdot \sum_{x_i < x} 1 \\
&= \frac{1}{n} \cdot |\{x_i : 1 \leq i \leq n \text{ and } x_i < x\}| \\
&\leq \frac{1}{n} \cdot \frac{n}{2} \\
&= \frac{1}{2},
\end{aligned}$$

and

$$\begin{aligned}
\sum_{x_i<x} w_i &= \sum_{x_i<x} \frac{1}{n} \\
&= \frac{1}{n} \cdot \sum_{x_i<x} 1 \\
&= \frac{1}{n} \cdot |\{x_i : 1 \le i \le n \text{ and } x_i > x\}| \\
&\le \frac{1}{n} \cdot \frac{n}{2} \\
&= \frac{1}{2},
\end{aligned}$$

which proves tat x is also the weighted median of $x_1, x_2, \dots, x_n$ with weights $w_i = 1/n$, for $i = 1, 2, \cdots, n$.

b. We first sort the n elements into increasing order by x_i values. Then we scan the array of sorted x_i's, starting with the smallest element and accumulating weights as we scan, until the total exceeds 1/2. The last element, say x_k, whose weight caused the total to exceed 1/2 is the weighted median. Notice that the total weight of all elements smaller than x_k is less than 1/2, because x_k was the first element that caused the total weight to exceed 1/2. Similarly, the total weight of all elements larger than x_k is also less than 1/2, because the total weight of all the other elements exceeds 1/2.

The sorting phase can be done in $O(n \lg n)$ worst-case time (using merge sort or heapsort), and the scanning phase takes $O(n)$ time. The total running time in the worst case, therefore, is $O(n \lg n)$.

c. We find the weighted median in $\Theta(n)$ worst-case time using the $\Theta(n)$ worst-case median algorithm in section 9.3.

The weighted-median algorithm works as follows. If $n \le 2$, we just return the brute-force solution. Otherwise, we proceed as follows. We find the actual median x_k of the n elements and then partition around it. We then compute the total weights of the two halves. If the weights of the two halves are each strictly less than 1/2, then the weighted median is x_k. Otherwise, the weighted median should be in the half with total weight exceeding 1/2. The total weight of the "light" half is lumped into the weight of x_k, and the search continues within the half that weighs more than 1/2. Here's pseudocode, which takes as input a set $X = \{x_1, x_2, ..., x_n\}$:

```
WEIGHTED-MEDIAN ($X$)
1. if $n = 1$
2.     then return $x_1$
3. elseif $n = 2$
4.     then if $w_1 \geq w_2$
5.         then return $x_1$
6.         else return $x_2$
7. else
8.     find the median $x_k$ of $X = \{x_1, x_2, ..., x_n\}$
9.     partition the set $X$ around $x_k$
10.    compute $W_L = \sum_{x_i < x_k} w_i$ and $W_G = \sum_{x_i > x_k} w_i$
11.    if $W_L < 1/2$ and $W_G < 1/2$
12         then return $x_k$
13.    elseif $W_L > 1/2$
14.        then $w_k \leftarrow w_k + W_G$
15.            $X' \leftarrow \{x_i \in X : x_i \leq x_k\}$
16.            return WEIGHTED-MEDIAN($X'$)
17.    else $w_k \leftarrow w_k + W_L$
18.            $X' \leftarrow \{x_i \in X : x_i \geq x_k\}$
16.            return WEIGHTED-MEDIAN($X'$)
```

The recurrence for the worst-case running time of WEIGHTED-MEDIAN is $T(n) = T(n/2 + 1) + \Theta(n)$, since there is at most one recursive call on half the number of elements, plus the median element x_k, and all the work preceding the recursive call takes $\Theta(n)$ time. The solution of the recurrence is $T(n) = \Theta(n)$.

Problems d. and e. are not representative of what will be on the mid-term. d.*The post-office location problem.*
Let the n points be denoted by their coordinates $x_1, x_2, \ldots, x_n$, let the corresponding weights be $w_1, w_2, \ldots, w_n$, and let $x = x_k$ be the weighted median. For any point p, let $f(p) = \sum_{i=1}^{n} w_i |p - x_i|$; we want to find a point p such that $f(p)$ is minimum. Let y be any point (real number) other than x. We show the optimality of the weighted median x by showing that $f(y) - f(x) \geq 0$. We examine separately the cases in which $y > x$ and $x > y$. For any x and y, we have

$$\begin{aligned} f(y) - f(x) &= \sum_{i=1}^{n} w_i |y - x_i| - \sum_{i=1}^{n} w_i |x - x_i| \\ &= \sum_{i=1}^{n} w_i (|y - x_i| - |x - x_i|). \end{aligned}$$

When $y > x$, we bound the quantity $|y - x_i| - |x - x_i|$ from below by examining three cases:
1. $x < y \leq x_i$: Here, $|x - y| + |y - x_i| = |x - x_i|$ and $|x - y| = y - x$,which imply that $|y - x_i| - |x - x_i| = -|x - y| = x - y$.

2. $x < x_i \leq y$: Here, $|y - x_i| \geq 0$ and $|x_i - x| \leq y - x$, which imply that $|y - x_i| - |x - x_i| \geq -(y - x) = x - y$.
3. $x_i \leq x < y$: Here, $|x - x_i| + |y - x| = |y - x_i|$ and $|y - x| = y - x$, which imply that $|y - x_i| - |x - x_i| = |y - x| = y - x$.
Separating out the first two cases, in which $x < x_i$, from the third case, in which $x \geq x_i$, we get

$$\begin{aligned} f(y) - f(x) &= \sum_{i=1}^{n} w_i(|y - x_i| - |x - x_i|) \\ &\geq \sum_{x<x_i} w_i(x - y) + \sum_{x \geq x_i} w_i(y - x) \\ &= (y - x)(\sum_{x \geq x_i} w_i - \sum_{x<x_i} w_i). \end{aligned}$$

The property that $\sum_{x_i<x} w_i < 1/2$ implies that $\sum_{x \geq x_i} w_i \geq 1/2$. This fact, combined with $y - x > 0$ and $\sum_{x<x_i} w_i \leq 1/2$, yields that $f(y) - f(x) \geq 0$.
When $x > y$, we again bound the quantity $|y - x_i| - |x - x_i|$ from below by examining three cases:
1. $x_i \leq y < x$: Here, $|y - x_i| + |x - y| = |x - x_i|$ and $|x - y| = x - y$, which imply that $|y - x_i| - |x - x_i| = -|x - y| = y - x$.
2. $y \leq x_i < x$: Here, $|y - x_i| \geq 0$ and $|x - x_i| \leq x - y$, which imply that $|y - x_i| - |x - x_i| \geq -(x - y) = y - x$.
3. $y < x \leq x_i$: Here, $|x - y| + |x - x_i| = |y - x_i|$ and $|x - y| = x - y$,which imply that $|y - x_i| - |x - x_i| = |x - y| = x - y$.
Separating out the first two cases, in which $x > x_i$, from the third case, in which $x \leq x_i$, we get

$$\begin{aligned} f(y) - f(x) &= \sum_{i=1}^{n} w_i(|y - x_i| - |x - x_i|) \\ &\geq \sum_{x>x_i} w_i(y - x) + \sum_{x \leq x_i} w_i(x - y) \\ &= (x - y)(\sum_{x \leq x_i} w_i - \sum_{x>x_i} w_i). \end{aligned}$$

The property that $\sum_{x_i>x} w_i \leq 1/2$ implies that $\sum_{x \leq x_i} w_i > 1/2$.This fact, combined with $x - y > 0$ and $\sum_{x>x_i} w_i < 1/2$, yields that $f(y) - f(x) > 0$.

e. We are given n 2-dimensional points $p_1, p_2, \ldots, p_n$, where each p_i is a pair of real numbers $p_i = (x_i, y_i)$, and positive weights $w_1, w_2, \ldots, w_n$. The goal is to find a point $p = (x, y)$ that minimizes the sum

$$f(x, y) = \sum_{i=1}^{n} w_i(|x - x_i| + |y - y_i|).$$

We can express the cost function of the two variables, $f(x,y)$, as the sum of two functions of one variable each: $f(x,y) = g(x) + h(y)$, where $g(x) = \sum_{i=1}^{n} w_i|x-x_i|$ and $h(y) = \sum_{i=1}^{n} w_i|y-y_i|$. The goal of finding a point $p = (x,y)$ that minimizes the value of $f(x,y)$ can be achieved by treating each dimension independently, because g does not depend on y and h does not depend on x. Thus,

$$\begin{aligned}
\min_{x,y} f(x,y) &= \min_{x,y}(g(x) + h(y)) \\
&= \min_{x}(\min_{y}(g(x) + h(y))) \\
&= \min_{x}(g(x) + \min_{y} h(y)) \\
&= \min_{x} g(x) + \min_{y} h(y).
\end{aligned}$$

Consequently, finding the best location in 2 dimensions can be done by finding the weighted median x_k of the x-coordinates and then finding the weighted median y_j of the y-coordinates. The point (x_k, y_j) is an optimal solution for the 2 dimensional post-office location problem.