Solutions to CS/MCS 401 Week #8–9 Exercises (Fall 2007)

Exercise J. Consider first the general case in which the pivot is chosen as the median of k elements, where k is odd and $k > 1$. We also assume $n >> k^2$, so we may approximate choice of k distinct elements by choice of k elements with repetition allowed..

Let $A_S = \{a[i] \mid a[i] < n/4^{\text{th}}$ smallest element of $a\}$, and let $A_L = \{a[i] \mid a[i] > n/4^{\text{th}}$ largest element of $a\}$. In general, the probability that among k randomly chosen elements of a exactly i lie in an $n/4$ element subset of a is very close to $C(k,i)(1/4)^i(3/4)^{k-i}$, since n is large. In order to obtain a bad split, at least $(k+1)/2$ elements from our k–element subset must lie in A_S, or at least $(k+1)/2$ must lie in A_L. Each of these alternatives occurs with probability

$$\sum_{i=(k+1)/2}^{k} C(k,i)(1/4)^i(3/4)^{k-i},$$

so the probability of a bad split is twice this, or

$$\sum_{i=(k+1)/2}^{k} 2C(k,i)(1/4)^i(3/4)^{k-i},$$

For $k = 1$, 3, 5, and 7, this evaluates as followed:

$k = 1$: $1/2 =$ **0.500**

$k = 3$: $2\cdot3(1/16)(3/4) + 2\cdot1(1/64) = 20/64 =$ **0.312**

$k = 5$: $2\cdot10(1/64)(9/16) + 2\cdot5(1/256)(3/4) + 2\cdot1\cdot(1/1024) = 212/1024 =$ **0.207**

$k = 7$: $2\cdot35(1/256)(27/64) + 2\cdot21(1/1024)(9/16) + 2\cdot7(1/4096)(3/4) + 2\cdot1(1/16384) =$ **0.139**.

Exercise 9.3-1. Let us consider the general case where the input elements are divided into groups of q elements each, where q is odd. For simplicity, assume n is a multiple of q, so the number of groups is n/q. Let $T_q(n)$ be the time to find k^{th} smallest element. In class and in the text, $q = 5$ was used. For a fixed q, the *Select*() algorithm uses constant time to sort (or at least to find the median of) each group of q elements, and hence time linear in n to sort all q-element groups. (However, the constant multiplying n does increase as q increases.) Then *Select*() invokes itself recursively to find the median of the n/q medians, requiring time $T(n/q)$. Next *Select*() invokes *partition*(), modified to used the median of the n/q medians as the pivot. This takes linear time, and produces a split in which at least $(q+1)/2 \cdot n/2q - 1$ of the n elements are less than the pivot, and at least this number are greater. Ignoring the -1, the fraction of elements on either side of the pivot is at least $(q+1)/(4q)$, meaning that the fraction on either side of the pivot can be at most $1 - (q+1)/(4q) = (3q-1)/(4q)$. The time for the recursive call to *Select*() on the left or right subarray is at most $T((3q-1)/(4q)\cdot n)$. So our recurrence for the running time is

$$T(n) = T(n/q) + T((3q-1)/(4q)\cdot n) + \Theta(n).$$

The sum of the subproblem sizes is $n/q + (3q-1)/(4q)\cdot n + \Theta(n) = 3(q+1)/4q \cdot n$. We showed in class that the solution was $\Theta(n)$ when $q = 5$; the proof relied only on the fact that the sum of the subproblem sizes was at most cn for some constant c less than 1. On the other

hand, if the subproblems have size αn and βn with $\alpha+\beta = 1$, the solution is $\Theta(n \lg(n))$. (We proved this in class for $\alpha = 2/3$, $\beta = 1/3$, but the proof works for any α and β with $\alpha < 1$, $\beta < 1$, and $\alpha+\beta = 1$. So for *Select*() to run in linear time in the worst case, we need $3(q+1)/4q < 1$, or $3(q+1) < 4q$, or $q > 3$.

Thus, with groups of 3, the worst-case running time of *Select*() is not linear, but with groups of 5, 7, or any other odd integer it is linear.

Exercise 9.3-8. First note: Let S be any set. If, for some q with $q < |S|/2$, we remove from S both q elements less than or equal to the (old) median of S and q elements greater than or equal to the median, then the median of S remains unchanged.

To keep things simple, let us assume $n = 2^k - 1$ for some k.

If $k = 1$, then we are finding the median of two elements; simply choose the smaller (for the lower median).

If $k > 1$, then $n \geq 3$. Let $m = (n+1)/2 = 2^{k-1}$, the middle position of X and Y. Since X and Y are sorted, the medians of X and Y are X[m] and Y[m], respectively. If X[m] = Y[m], we are done; X[m] is the median of the two arrays combined. If X[m] < Y[m], then

$$X[m] \leq \text{(combined median)} \leq Y[m].$$

We may discard X[1..m] (m elements ≤ combined median) and Y[m..n] (m elements ≥ the combined median). This leaves two sorted subarrays, X[m+1..n] and Y[1..m−1] with $2^{k-1} - 1$ each, which have the same combined median as the two original arrays, so we can invoke our algorithm recursively to find the median of these two subarrays. Similarly, if X[m] > Y[m], simply reverse the roles of X and Y in the previous argument.

```
// Invoke initially as median( X, 1, n, Y, 1, n). T is the element type of X and Y. Note
//  this code assumes for simplicity that n is one less than a power of 2. Note at all times
//  xRight−xLeft = yRight−yLeft.

T median( T[] X, int xLeft, int xRight, T[] Y, int yLeft, int yRight)
    if ( xLeft == xRight )
        return min( X[xLeft], Y[yLeft] );
    xMid = (xLeft + xRight) / 2;
    yMid = (yLeft + yRight) / 2;
    if ( X[xMid] < Y[yMid] )
        return median( X, xMid+1, xRight, Y, yLeft, yMid−1);
    else if ( X[xMid] > Y[yMid] )
        return median( X, xLeft, xMid−1, Y, yMid+1, yRight);
    else
        return X[xMid];
```