Problems Chapter 9

Exercise 9.2-1

Show that RANDOMIZED-SELECT never makes a recursive call to a 0-length array.

Assume, to the contrary, that RANDOMIZED-SELECT does make a recursive call to a 0-length array. Then, this recursive call must occur on line 8 or 9. If it occurs on line 8, then it must be the case that $q = p$ after the call to RANDOMIZED-PARTITION on line 3. In which case, $k = 1$ and $i \neq 1$, since otherwise we would have returned $A[q]$ on line 6. But since, $k = 1 \neq i$, it cannot be the case that $i < k$ and so the recursive call to RANDOMIZED-SELEECT on line 8 is not executed. But this contradicts our assumption. Otherwise, the recursive call to RANDOMIZED-SELECT to a 0-length array must occur on line 9. In which case we must have $q = r$. By similar reasoning as above, however, if $q = r$, then we must have $i < k$ and so the recursive call on line 9 cannot possibly execute. Therefore, RANDOMIZED-SELECT does not make a recursive call to a 0-length array.

Exercise 9.2-2

Argue that the indicator random variable X_k and the value $T(max(k-1, n-k))$ are independent.

Consider the the expression $max(k-1, n-k)$ which is defined as

$$max(k-1, n-k) = \begin{cases} k-1 & \text{if } k > \lceil n/2 \rceil \\ n-k & \text{if } k \leq \lceil n/2 \rceil \end{cases} \tag{1}$$

and let Y be the random variable that is the value of $max(k-1, n-k)$. Observe, that $P\{Y = k-1\} = P\{k > \lceil n/2 \rceil\} = \frac{n-\lceil n/2 \rceil}{n}$. This probability is independent of the value of k. And since $P\{Y = n-k\} = 1 - P\{Y = k-1\}$ we conclude that the random variable Y is independent of the value of k for all k. And since X_k is entirely dependent on this quantity, it follows that Y is independent of X_k and finally that the random variables X_k and the value $T(max(k-1, n-k))$ are independent.

Exercise 9.3-2

Analyze SELECT to show that if $n \geq 140$, then at least $\lceil n/4 \rceil$ elements are greater than the median-of-medians x and at least $\lceil n/4 \rceil$ elements are less than

x.

The analysis in the text has shown that for any median-of-medians, x, and sufficiently large n, the number of elements greater than x is at least $\frac{3n}{10} - 6$. The same analysis, except considering the elements that are less than x, will show that the lower bound for the number of elements less than x is the same. Using this lower bound, we may write the following inequality

$$\lceil n/4 \rceil \leq \frac{n}{4} + 1 \leq \frac{3n}{10} - 6 \tag{2}$$

whose solution is $n \geq 140$.

Exercise 9.3-3

Show how quicksort can be made to run in $O(n \lg n)$ time in the worst case, assuming that all elements are distinct.

The worst case running time of quicksort of $\Theta(n^2)$ occurs when the pivot element in the call to PARTITION is either the smallest element in the array, or the greatest element in the array. In other words, the worst case occurs when the pivot is either the 1st order statistic, or it is the nth order statistic. By using the technique of finding the median-of-medians introduced in this chapter, we can guarantee that PARTITION will use a good pivot.

We change the call to PARTITION to choose the pivot by first calling SELECT on the lower median of the array, $\lceil (r-p)/2 \rceil$. The call to select occurs only once, and takes time $O(n)$. Thus, the overall running time of partition of $\Theta(n)$ is unchanged. Thus, the sizes of the subproblems produced by PARTITION are $O(n/2)$. Therefore, we may bound the resulting recurrence relation as follows

$$T(n) \leq 2T(n/2) + O(n) \tag{3}$$

whose solution, by the master theorem, is $T(n) = O(n \lg n)$.

Exercise 9.3-4

Suppose that an algorithm uses only comparisons to find the ith smallest element in a set of n elements. Show that it can also find the $i-1$ smaller elements and the $n-i$ larger elements without performing any additional comparisons.

In order to find the ith smallest element, the algorithm must partition the elements into the $i-1$ that are less than the ith smallest element, and the $n-i$ that are greater. Since the algorithm only uses comparisons, it must, in the process of partitioning the elements, identify each one. By keeping track of which partition each element is assigned as they are assigned, the algorithm can find the $i-1$ smaller elements and the $n-i$ larger elements without performing any additional comparisons.

Exercise 9.3-5

Suppose that you have a "black-box" worst-case linear-time median subroutine. Give a simple linear-time algorithm that solves the selection problem for an arbitrary order statistic.

Let A be the input array of n elements, k be the median of n and i be the order statistic we wish to find. The algorithm is as follows. Run the linear-time median subroutine on the input. If $i == k$, then we are done. Otherwise, if $i < k$, discard those elements of A that are greater than or equal to k and recursively call this algorithm on the remaining elements. If $i > k$, discard those elements of A that are less than or equal to k and recursively call the algorithm on the remaining elements, setting $i = i - k$. Assuming that the elements of the array are distinct, each iteration will result in a subproblem of size that is $O(n/2)$. Additionally, the filtering can be done in lienear time. Therefore, the recurrence relation of this algorithm can be bounded by

$$T(n) \leq T(n/2) + O(n) \tag{4}$$

and the solution of this recurrence, by the master theorem, is $T(n) = O(n)$.

Exercise 9.3-6

The kth ***quantiles*** of an n-element set are the $k-1$ order statistics that divide the sorted set into k equal-sized sets (to within 1). Give an $O(n \lg k)$-time algorithm to list the kth quantiles of a set.

Let m_k be the median of the elements in the kth quantile. We can determine the order statistic with respect to n of m_k in constant time. This is simply $o_n(m_k) = \lceil \frac{k-1}{2} \rceil \lceil \frac{n}{k} \rceil$. We can use this as the basis of the desired algorithm, call it KQUANTILE. Call SELECT on the input array A with $i = o_n(m_k)$. Store the element that is returned, and then recursively call KQUANTILE on the subarray $A[1..o_n(m_k) - 1]$ with a new $k = \lceil \frac{k-1}{2} \rceil$. And recursively call KQUANTILE on the subarray $A[o_n(m_k) + 1..n]$ with a new k defined as:

$$k = \begin{cases} \lceil \frac{k-1}{2} \rceil & \text{if } k \equiv 0 \mod 2 \\ \lceil \frac{k-1}{2} \rceil + 1 & \text{if } k \equiv 1 \mod 2 \end{cases}$$

Each recursive call of KQUANTILE will work on the entire array and so run in time $O(n)$. But each level of recursion will find 2 times as many of the elements in the kth quantile as the last. Consequently, there will be at most $O(\lg k)$ recursive calls. Hence the running time of KQUANTILE is $O(n \lg k)$.

Exercise 9.3-7

Describe an $O(n)$-time algorithm that, given a set S of n distinct numbers and a positive integer $k \leq n$, determines the k numbers in S that are closest to the median of S.

Let m be the order statistic of the median. Use SELECT to find the median and

the $m-k$ and $m+k$ order statistics of S. Do a linear scan over these elements computing the absolute value of their difference with the median. Initialize pointers to the $m+1$ and $m-1$ order statistics. Compare them, take the lesser, and move its pointer to the next element. Repeat until the k closest elements have been found. The m, $m+k$th and $m-k$th order statistics can be found in linear time. The absolute value of the difference between the elements and the median may also be found in time that is $O(n)$, since $k = O(n)$, and the final comparison and selection of elements can also be done in time $O(n)$. Therefore the algorithm runs in $O(n)$-time.

Exercise 9.3-8

Let $X[1..n]$ and $Y[1..n]$ be two arrays, each containing n numbers already in sorted order. Give an $O(\lg n)$-time algorithm to find the median of all $2n$ elements in arrays X and Y.

The big idea, is that on each recursive call, we can eliminate half of the elements

Algorithm 1 Algorithm for finding the ith order statistic of all elements in arrays X and Y, assuming that X and Y are already sorted. If A is an array, then $A.n$ is the number of elements in the array, m_A is the median of the elements in array A and $o_A(m_A)$ is the associated order statistic of the median of A in A.

```
function SELECTMOD(X, Y, i)
    if X.n = 0 then return Y[i]
    if Y.n = 0 then return X[i]
    m_X = median of X
    m_Y = median of Y
    if m_X < m_Y then
        if i < o_X(m_x) + o_Y(m_y) then
            q = SELECTMOD(X, Y[1..o_Y(m_Y) − 1], i)
        else
            q = SELECTMOD(X[o_X(m_X) + 1..X.n], Y, i − o_X(m_X))
    else
        if i < o_X(m_X) + o_Y(m_Y) then
            q = SELECTMOD(X[1..o_X(m_X) − 1], Y, i)
        else
            q = SELECTMOD(X, Y[o_Y(m_Y) + 1..Y.n], i − o_Y(M_))
    return q
```

of one of the remaining arrays from consideration by comparing the medians of the arrays. If the median of X is less than the median of Y, then it follows that all the elements of X less than or equal to its median have order statistics in the aggregated array less than the sum of the orders of the medians. Therefore, if the order statistic of the median of the aggregated array is greater than this sum, we may discard these elements in X and repeat with fewer elements. This will take at most $\lg(X.n) + \lg(Y.n)$ recursive calls with constant work done on each. In the case where $X.n = Y.n = n$, therefore, the algorithm runs in time $O(\lg(n))$.

Exercise 9.3-9

Professor Olay is consulting for an oil company, whcih is planning a large pipeline running east to west through an oil field of n wells. The company wants to connect a spur pipeline from each well directly to the main pipeline along a shortest route (either north or south), as shown in Figure 9.2. Given the x- and y-coordinates of the wells, how should the professor pick the optimal location of the main pipeline, which would be the one that minimizes the total length of the spurs? Show how to determine the optimal location in linear time.

Find the median oil well y-coordinate. If n is odd, run the main-pipeline along that y-coordinate. If n is odd, run the main-pipeline along the y-coordinate that is halfway between the y-coordinates of the lower and upper medians. Since we can find all the median y-coordinates in linear time, and the rest of this solution requires constant time, the overall solution running time is linear.

Problem 9-1 Largest i numbers in sorted order

Given a set of n numbers, we wish to find the i largest in sorted order using a comparison-based algorithm. Find the algorithm that implements each of the following methods with the best asymptotic worst-case runnign time, and analyze the running times of the algorithms in terms of n and i.

a. Sort the numbers, and list the i largest.

Use heapsort, but instead of extracting the maximum element n times we would need only extract it i times. The running time would therefore be $O(n \lg n)$ since $i = O(n)$.

b. Build a max-priority queue from the numbers, and call EXTRACT-MAX i times.

Same as above.

c. Use an order-statistic algorithm to find the ith largest number, partition around that number, and sort the i largest numbers.

Use SELECT to find the ith largest number and partition around it in $O(n)$ time. Then use heapsort or mergesort on the $n - i$ largest numbers. This will run in time $O(n + i\text{lg}(i))$.

9-2 Weighted median

For n distinct elements $x_1, x_2, ..., x_n$ with positive weights $w_1, w_2, ..., w_n$ such that $\Sigma_{n=1}^{n} w_i = 1$, the ***weighted (lower) median*** is the elements x_k satisfying

$$\sum_{x_i < x_k} w_i < \frac{1}{2}$$

and

$$\sum_{x_i > x_k} w_i \leq \frac{1}{2}$$

For example, if the elements are 0.1,0.35,0.05,0.1,0.15,0.05,0.2 and each element equals its weight (that is, $w_i = x_i$ for $i = 1, 2, ..., 7$), then the median is 0.1, but the weighted median is 0.2.

a. Argue that the median of $x_1, x_2, ...x_n$ is the weighted median of the x_i with weights $w_i = 1/n$ for $i = 1, 2, ..., n$.

Let x_m be the median of the x_i. If n is even, then there are $n/2 - 1$ elements less than x_m and so $\Sigma_{x_i < x_m} w_i = \frac{n/2-1}{n} < \frac{1}{2}$ and there are $n/2$ elements greater than x_m and so $\Sigma_{x_i > x_m} w_i = \frac{n/2}{n} = \frac{1}{2}$. Otherwise, n is odd, in which case there are fewer than $n/2$ elements less than, or greater than x_m and so $\Sigma_{x_i < x_m} w_i < \frac{n/2}{n} < \frac{1}{2} \geq \Sigma_{x_i > x_m} w_i$ and so we conclude that the median x_m is the weighted median.

b. Show how to compute the weighted median of n elements in $O(n\lg n)$ worst-case time using sorting.

Use mergesort or heapsort to sort the elements on their weights. Starting with the least weighted element, sum the weights until the sum is greater than or equal to 1/2. The element whose weight was added last is the weighted median. The sorting will run in time $O(n\lg n)$ and the accumulation will take time $O(n)$, hence the overall running time is $O(n\lg n)$.

c. Show how to compute the weighted median in $\Theta(n)$ worst-case time using a linear-time median algorithm such as SELECT from section 9.3.

This one stumped me. I had to look up the solution.