Chapter 9

Michelle Bodnar, Andrew Lohr

April 12, 2016

Exercise 9.1-1

In this problem, we will be recursing by dividing the array into two equal size sets of elements, we will neglect taking floors and ceilings. The analysis will be the same but just a bit uglier if we don't assume that n is a power of 2.

Break up the elements into disjoint pairs. Then, compare each pair, consider only the smallest elements from each. From among this set of elements, the result to the original problem will be either what had been paired with the smallest element, or what what is the second smallest element of the sub-problem. Doing this will get both the smallest and second smallest element. So, we get the recurrence $T(n) = T(n/2)+n/2+1$ and $T(2) = 1$. So, solving this recurrence, we will use the substitution method with $T(n) \leq n + \lceil \lg(n) \rceil - 2$, it agrees with the base case, and, $T(n) = n/2+T(n/2)+1 \leq n/2+n/2+\lceil \lg(n/2) \rceil -2+1 = n+\lceil \lg(n) \rceil -2$ as desired.

Exercise 9.1-2

Initially, all n numbers are potentially either the maximum or minimum. Let MAX be the set of numbers which are potentially the maximum and MIN denote the set of numbers which are potentially the minimum. Say we compare two elements a and b. If $a \leq b$, we can remove a from MAX and remove b from MIN, so we reduce the counts of both sets by 1. If we compare two elements in MIN, we reduce the size of MIN by 1, and if we compare two elements in MAX, we can reduce the size of MAX by 1. Thus, the first type of comparison is optimal. There are $\lceil n/2 \rceil$ such comparisons which can be made until MIN and MAX are disjoint, and the sets will have sizes $\lceil n/2 \rceil$ and $\lfloor n/2 \rfloor$. Within each of these, only the second and third types of comparisons can be made, each reducing the set size by 1, so we require a total of $\lceil n/2 \rceil - 1 + \lfloor n/2 \rfloor - 1 = n - 2$ comparisons. Adding this to initial $\lceil n/2 \rceil$ comparisons gives $\lceil 3n/2 \rceil - 2$.

Exercise 9.2-1

Calling a zero length array would mean that the second and third arguments are equal. So, if the call is made on line 8, we would need that $p = q - 1$. which

means that $q - p + 1 = 0$ .However, i is assumed to be a nonnegative number, and to be executing line 8, we would need that $i < k = q - p + 1 = 0$, a contradiction. The other possibility is that the bad recursive call occurs on line 9. This would mean that $q + 1 = r$. To be executing line 9, we need that $i > k = q - p + 1 = r - p$. This would be a nonsensical original call to the array though because we are asking for the ith element from an array of strictly less size.

Exercise 9.2-2

The probability that X_k is equal to 1 is unchanged when we know the max of $k-1$ and $n-k$. In other words, $P(X_k = a| \max(k-1, n-k) = m) = P(X_k = a)$ for $a = 0, 1$ and $m = k - 1, n - k$ so X_k and $\max(k - 1, n - k)$ are independent. By C.3-5, so are X_k and $T(\max(k - 1, n - k))$.

Exercise 9.2-3

Algorithm 1 ITERATIVE-RANDOMIZED-SELECT

```
while p < r do
    q = RANDOMIZED − PARTITION(A, p, r)
    k = q − p + 1
    if i=k then
        return A[q]
    end if
    if i < k then
        r = q − 1
    else
        p = q
        i = i − k
    end if
end while
return A[p]
```

Exercise 9.2-4

When the partition selected is always the maximum element of the array we get worst-case performance. In the example, the sequence would be 9, 8, 7, 6, 5, 4, 3, 2, 1, 0.

Exercise 9.3-1

It will still work if they are divided into groups of 7, because we will still know that the median of medians is less than at least 4 elements from half of the $\lceil n/7 \rceil$ groups, so, it is greater than roughly $4n/14$ of the elements. Similarly, it

is less than roughly $4n/14$ of the elements. So, we are never calling it recursively on more than $10n/14$ elements. So, $T(n) \leq T(n/7) + T(10n/14) + O(n)$. So, we can show by substitution this is linear. Suppose $T(n) < cn$ for $n < k$, then, for $m \geq k$, $T(m) \leq T(m/7) + T(10m/14) + O(m) \leq cm(1/7 + 10/14) + O(m)$. So, as long as we have that the constant hidden in the big-Oh notation is less than $c/7$, we have the desired result.

Suppose now that we use groups of size 3 instead. So, For similar reasons, we have that the recurrence we are able to get is $T(n) = T(\lceil n/3 \rceil) + T(4n/6) + O(n) \geq T(n/3) + T(2n/3) + O(n)$. So, we will show it is $\geq cn \lg(n)$.

$T(m) \geq c(m/3)\lg(m/3) + c(2m/3)\lg(2m/3) + O(m) \geq cm\lg(m) + O(m)$. So, we have that it grows more quickly than linear.

Exercise 9.3-2

We know that the number of elements greater than or equal to x and the number of elements less than or equal to x is at least $3n/10 - 6$. Then for $n \geq 140$ we have

$$3n/10 - 6 = \frac{n}{4} + \frac{n}{20} - 6 \geq n/4 + 140/20 - 6 = n/4 + 1 \geq \lceil n/4 \rceil.$$

Exercise 9.3-3

We can modify quicksort to run in worst case $n\lg(n)$ time by choosing our pivot element to be the exact median by using quick select. Then, we are guaranteed that our pivot will be good, and the time taken to find the median is on the same order of the rest of the partitioning.

Exercise 9.3-4

Create a graph with n vertices and draw a directed edge from vertex i to vertex j if the i^{th} and j^{th} elements of the array are compared in the algorithm and we discover that $A[i] \geq A[j]$. Observe that $A[i]$ is one of the $i - 1$ smaller elements if there exists a path from x to i in the graph, and $A[i]$ is one of the $n - i$ larger elements if there exists a path from i to x in the graph. Every vertex i must either lie on a path to or from x because otherwise the algorithm can't distinguish between $i \leq x$ and $i \geq x$. Moreover, if a vertex i lies on both a path to x and a path from x then it must be such that $x \leq A[i] \leq x$, so $x = A[i]$. In this case, we can break ties arbitrarily.

Exercise 9.3-5

To use it, just find the median, partition the array based on that median. If i is less than half the length of the original array, recurse on the first half, if i is half the length of the array, return the element coming from the median finding

black box. Lastly, if i is more than half the length of the array, subtract half the length of the array, and then recurse on the second half of the array.

Exercise 9.3-6

Without loss of generality assume that n and k are powers of 2. We first find the $n/2^{th}$ order statistic, in time $O(n)$ using SELECT, then reduce the problem to finding the $k/2^{th}$ quantiles of the smaller $n/2$ elements and the $k/2^{th}$ quantiles of the larger $n/2$ elements. Let $T(n)$ denote the time it takes the algorithm to run on input of size n. Then $T(n) = cn + 2T(n/2)$ for some constant c, and the base case is $T(n/k) = O(1)$. Then we have:

$$\begin{aligned} T(n) &\leq cn + 2T(n/2) \\ &\leq 2cn + 4T(n/4) \\ &\leq 3cn + 8T(n/8) \\ &\vdots \\ &\leq \log(k)cn + kT(n/k) \\ &\leq \log(k)cn + O(k) \\ &= O(n \log k). \end{aligned}$$

Exercise 9.3-7

Find the $n/2 - k/2$ largest element in linear time. Partition on that element. Then, find the k largest element in the bigger subarray formed from the partition. Then, the elements in the smaller subarray from partitioning on this element are the desired k numbers.

Exercise 9.3-8

Without loss of generality, assume n is a power of 2.

Algorithm 2 Median(X,Y,n)

```
if n==1 then
    return min(X[1], Y[1])
end if
if X[n/2] < Y[n/2] then
    return Median(X[n/2 + 1...n], Y[1...n/2], n/2)
else
    if X[n/2] ≥ Y[n/2] then
        return Median(X[1...n/2], Y[n/2 + 1...n], n/2)
    end if
end if
```

Exercise 9.3-9

If n is odd, then, we pick the y coordinate of the main pipeline to be equal to the median of all the y coordinates of the wells. if n is even, then, we can pick the y coordinate of the pipeline to be anything between the y coordinates of the wells with y-coordinates which have order statistics $\lfloor (n+1)/2 \rfloor$ and the $\lceil (n+1)/2 \rceil$. These can all be found in linear time using the algorithm from this section.

Problem 9-1

a. Sorting takes time $n \lg(n)$, and listing them out takes time i, so the total runtime is $O(n \lg(n) + i)$

b. Heapifying takes time $n \lg(n)$, and each extraction can take time $\lg(n)$, so, the total runtime is $O((n+i) \lg(n))$

c. Finding and partitioning around the ith largest takes time n. Then, sorting the subarray of length i coming from the partition takes time $i \lg(i)$. So, the total runtime is $O(n + i \lg(i))$.

Problem 9-2

a. Let m_k be the number of x_i smaller than x_k. When weights of $1/n$ are assigned to each x_i, we have $\sum_{x_i<x_k} w_i = m_k/n$ and $\sum_{x_i>x_k} w_i = (n - m_k - 1)/2$. The only value of m_k which makes these sums $< 1/2$ and $\leq 1/2$ respectively is when $\lceil n/2 \rceil - 1$, and this value of x must be the median since it has equal numbers of $x_i's$ which are larger and smaller than it.

b. First use mergesort to sort the x_i's by value in $O(n \log n)$ time. Let S_i be the sum of the weights of the first i elements of this sorted array and note that it is $O(1)$ to update S_i. Compute $S_1, S_2, \ldots$ until you reach k such that $S_{k-1} < 1/2$ and $S_k \geq 1/2$. The weighted median is x_k.

c. We modify SELECT to do this in linear time. Let x be the median of medians. Compute $\sum_{x_i<x} w_i$ and $\sum_{x_i>x} w_i$ and check if either of these is larger than $1/2$. If not, stop. If so, recurse on the collection of smaller or larger elements known to contain the weighted median. This doesn't change the runtime, so it is $\Theta(n)$.

d. Let p be the minimizer, and suppose that p is not the weighted median. Let ϵ be small enough such that $\epsilon < \min_i(|p - p_i|)$, where we don't include k if $p = p_k$. If p_m is the weighted median and $p < p_m$, choose $\epsilon > 0$. Otherwise

choose $\epsilon < 0$. Then we have

$$\sum_{i=1}^{n} w_i d(p+\epsilon, p_i) = \sum_{i=1}^{n} w_i d(p, p_i) + \epsilon \left(\sum_{p_i < p} w_i - \sum_{p_i > p} w_i \right) < \sum_{i=1}^{n} w_i d(p, p_i)$$

since the difference in sums will take the opposite sign of epsilon.

e. Observe that

$$\sum_{i=1}^{n} w_i d(p, p_i) = \sum_{i=1}^{n} w_i |p_x - (p_i)_x| + \sum_{i=1}^{n} w_i |p_y - (p_i)_y|.$$

It will suffice to minimize each sum separately, which we can do since we choose p_x and p_y individually. By part e, we simply take $p = (p_x, p_y)$ to be such that p_x is the weighted median of the x-coordinates of the p_i's and p_y is the weighted medain of the y-coordiantes of the p_i's.

Problem 9-3

a. If $i \geq n/2$, then just use the algorithm from this chapter to get the answer in time $T(n)$. If $i < n/2$, then, we can compare disjoint pairs of elements from the list, and then we know that the ith smallest is in the set of elements that are smaller in each pair. So, we can recurse, this gets us runtime in this case of $\lfloor n/2 \rfloor + U_i(\lceil n/2 \rceil) + T(2i)$. Note that the last term comes from the fact that the ith smallest could of also been any of the elements paired with the ith smallest elements from the subproblem.

b. By the Substitution method, suppose that $U_i(n) = n + cT(2i)\lg(n/i)$ for smaller n, then, there are two cases based on whether or not $i < n/4$. If it is, $U_i(n) = \lfloor n/2 \rfloor + U_i(\lceil n/2 \rceil) + T(2i) \leq n/2 + n/2 + cT(2i)\lg(n/2i) + T(2i)$. This then satisfies the recurrence if we have that $c \geq 1$. The other case is that $n/4 \leq i < n/2$. In this case, we have that $U_i(n) = n/2 + T(\lceil n/2 \rceil) + T(2i) \leq n/2 + 2T(2i)$, which works if we have $c \geq 2$. So, we can just pick $c = 2$, and both cases of the recurrence go through.

c. From the previous part, if i is a constant, the $O(T(2i)\lg(n/i))$ becomes $T(\lg(n))$. So, $U_i(n) = n + O(T(2i)\lg(n/i)) = n + O(\lg(n))$.

d. From part c, we just substitute in n/k for i to get $U_i(n) = n + O(T(2i)\lg(n/i)) = n + O(T(2n/k)\lg k)$.

Problem 9-4

a. We only need to worry about what happens when we select a pivot element which is between $\min(z_i, z_j, z_k)$ and $\max(z_i, z_j, z_k)$, since at this point we will either select z_i or z_j and compare them, select an element between z_i and z_j and never compare them, or select an element between z_k and the $[z_i, z_j]$ interval, so that we will will never again be selecting pivots from a range containing z_i and z_j. We split into three cases. If $z_k \leq z_i < z_j$ then $E(X_{ijk}) = \frac{2}{j-k+1}$. If $z_i \leq z_k < z_j$ then $E(X_{ijk}) = \frac{2}{j-i+1}$. If $z_i < z_j \leq z_k$ then $E(X_{ijk}) = \frac{2}{k-i+1}$.

b.

$$\begin{aligned} E[X_k] &\leq \sum_{i=1}^{n}\sum_{j=1}^{n} E[X_{ijk}] \\ &= \sum_{i=1}^{k}\sum_{j=k}^{n} \frac{2}{j-i+1} + \sum_{i=k+1}^{n}\sum_{j=i+1}^{n} \frac{2}{j-k+1} + \sum_{i=1}^{k-2}\sum_{j=i+1}^{k-1} \frac{2}{k-i+1}. \end{aligned}$$

For the second term, fix some value m. The term $\frac{2}{m-k+1}$ will appear once for every time j doesn't exceed m, so $m-(k+1)$ times in the sum. For the third term, each term in the sum doesn't depend on j so we can rewrite it as $2\sum_{i=1}^{k-2} \frac{k-i-1}{k-i+1}$. This gives

$$E[X_k] = 2\left(\sum_{i=1}^{k}\sum_{j=k}^{n} \frac{1}{j-i+1} + \sum_{j=k+1}^{n} \frac{j-k-1}{j-k+1} + \sum_{i=1}^{k-2} \frac{k-i-1}{k-i+1}\right).$$

c. We can bound the summands of the second and third terms from (b) by 1, so the total contribution of these terms is $n-(k+1)+1+k-2 = n-2$. For the first double sum, consider a term of the form $\frac{1}{c}$ for some c. There are at most c of these, because we must have $j-i = c-1$. The largest term which appears is 1 and the smallest term which appears is $\frac{1}{n}$, so the double sum contributes a total of at most n. Thus, $E[X_k] \leq 2(n+n-2) \leq 4n$.

d. The running time of RANDOMIZED-SELECT is dominated by the time spent in RANDOMIZED-PARTITION, whose running time is dominated by comparisons. By part (c) we know that the expected number of comparisons over the entire algorithm, including recursions, is $O(n)$.