pset3 #3

Alan Hui (binghui2), Caleb Ju (calebju2), Alex Gao (yougao2)

A Psuedocode:

Algorithm 1 find-k

Input list A of size n, a rank k
Output element of rank k: a

$i_0 \leftarrow \text{random}(1, n)$
$B, C \leftarrow$ partition of elements smaller and greater than A_{i_0} respectively
if $|B| = k - 1$ **then**
    $a \leftarrow A_{i_0}$
else if $|B| \geq k$ **then**
    $a \leftarrow \text{FIND-}k(B, k)$
else
    $a \leftarrow \text{FIND-}k(C, k - (|B| + 1))$
return a

Define the algorithm with the recursive depth of $D(n, k)$, where n is the size of the input and k is the rank we are looking for. From the algorithm, we have recurrance,

$$D(n,k) = \begin{cases} 0, \text{if } n < k \text{ or } k < 1 \text{ or } i = k, \\ 1 + T(i-1, k) + T(n-i, k-i), \text{elsewise}, \end{cases}$$

where $i = \text{rank}(A_{i_0})$. In other words, if either there aren't enough elements to find a rank k element or if the rank is an invalid index or we have found the right element, then we stop recursing. Otherwise, we "make" two recursive calls - in reality, only one call is actually made since one of the calls must be invalid due to one of the three stopping rules. Next, we find the expected value of $D(n, k)$ for a fixed k.

$$\begin{aligned}
\mathbb{E}[D(n,k)] &\leq 1 + \mathbb{E}[D(i-1,k)] + \mathbb{E}[D(n-i,k-i)] \\
&= 1 + \sum_{i=1}^{n} \mathbb{E}[D(i-1,k) | \text{rank}(a_{i_0}) = i] \cdot P[\text{rank}(a_{i_0}) = i] \\
&+ \sum_{i=1}^{n} \mathbb{E}[D(n-i,k-i) | \text{rank}(a_{i_0}) = i] \cdot P[\text{rank}(a_{i_0}) = i] \\
&= 1 + \frac{1}{n}\Big(\sum_{i=1}^{n} D(i-1,k) + D(n-i,k-i) \Big) \\
&= 1 + \frac{1}{n}\Big(\sum_{i=k+1}^{n} D(i-1,k) + \sum_{j=1}^{k-1} D(n-j,k-j) \Big).
\end{aligned}$$

The transition from line four to five is removing indicies from the summation that trivally have a depth of 0.

Similar to pset0 recurrance problem, we claim that $D(n,k) \le \alpha \log(n) + \beta$. The bound trivially holds for $D(1,k) = 1$ with arbitrary α and $\beta = 1$. To solve by induction for a $n > 1$, we have that

$$D(n,k) \le 1 + \frac{1}{n}\Big(\sum_{i=k+1}^{n} D(i-1,k) + \sum_{j=1}^{k-1} D(n-j,k-j)\Big)$$
$$\le 1 + \frac{1}{n}\Big(\frac{n}{2}(\alpha\log(n) + \beta) + \frac{n}{2}(\alpha(\log(n) - \log(4/3)) + \beta)\Big)$$
$$\text{which we wish to show is} \le \alpha\log(n) + \beta.$$

We see the recurrance and base case then hold for $\beta \ge 1$ and $\alpha \ge \frac{2+2\beta}{\log(4/3)}$, and thus completes the proof that $D(n,k) = O(\log(n))$.

Correctness is clear, but we can describe the proof, which is inductive. The case where the total number of elements is $n = 1$ is just returning the element of the list. Suppose the algorithm computes the correct element for sizes $< n$. Consider the n size case. If we randomly pick the right element, which means there are $k-1$ elements smaller than it, then we are done. If not, we either recurse on partition A or B, both of which must be at most size $n-1$ since we removed the partition element. If we recurse on A, by the inductive hypothesis we will compute the correct rank. If we recurse on B, the $k-i$th rank element of B will be correctly returned. This is also the kth rank item overall since it is also larger than the i elements of A and a_{i_0} $\square$

B We sketch the $O(h\lg^2 n)$ algorithm, which is nearly identical to part 1. Pick an arbitrary element x from any of the h sorted arrays. Then, split all h sorted groups into the partitions A_i and B_i with elements smaller and greater than x respectively. This can be done with binary search. If $\sum |A_i| > k-1$, then we recurse on the list of A_i. If the sum is equal to $k-1$, then we know x is the correct element, and the last case is recursing with B_i and changing $k = k - (1 + \sum |A_i|)$.

Notice the this problem is simply a reduction to part 1; the difference is the n items are separated into the h groups. In each level, instead of doing $O(1)$ work like in problem one, we now do $O(h \cdot \log(n))$ work from the h binary searches. So, if we replace the 1 in the expected value analysis with $c \cdot h\log n$ for some constant c, solving will lead to $D(n,k) = O(h\lg^2 n)$, where $D(n,k)$ is the work done of this algorithm.

Correctness follows by the proof of correctness for part 1.