CS3383, Winter 2019 Assignment # 3 Sample solutions

Rasoul Shahsavarifar
Faculty of Computer Science, UNB

Questions 1, 2, & 3: For the solutions, see your notes from tutorial on Wednesday, 6/Feb/2019.

Bonus Question) Partial solution:

Suppose we are searching for the k^{th} smallest element s_k in the union of the lists $a[1, \ldots, m]$ and $b[1, \ldots, n]$. Because we are searching for the k^{th} smallest element, we can restrict our attention to the arrays $a[1, \ldots, k]$ and $b[1, \ldots, k]$. If $k > m$ or $k > n$, we can take all the elements with index larger than the array boundary to have infinite value. Our algorithm starts off by comparing elements $a[\lfloor k/2 \rfloor]$ and $b[\lceil k/2 \rceil]$. Suppose $a[\lfloor k/2 \rfloor] > b[\lceil k/2 \rceil]$. Then, in the union of a and b there can be at most $(k-2)$ elements smaller than $b[\lceil k/2 \rceil]$, i.e. $a[1, \ldots, \lfloor k/2 \rfloor - 1]$ and $b[1, \ldots, \lceil k/2 \rceil - 1]$, and we must necessarily have $s_k > b[\lceil k/2 \rceil]$. Similarly, all elements $a[1, \ldots, \lfloor k/2 \rfloor]$ and $b[1, \ldots, \lceil k/2 \rceil]$ will be smaller than $a[\lfloor k/2 \rfloor + 1]$; but these are k elements, so we must have $s_k < a[\lfloor k/2 \rfloor + 1]$. This shows that s_k must be contained in the union of the subarrays $a[1, \ldots, \lfloor k/2 \rfloor]$ and $b[\lceil k/2 \rceil + 1, k]$. In particular, because we discarded $\lceil k/2 \rceil$ elements smaller than s_k, s_k will be the $\lfloor k/2 \rfloor^{th}$ smallest element in this union. We can then find s_k by recursing on this smaller problem. The case for $a[\lfloor k/2 \rfloor] < b[\lceil k/2 \rceil]$ is symmetric. The last case, which is also the base case of the recursion, is $a[\lfloor k/2 \rfloor] = b[\lceil k/2 \rceil]$, for which we have $s_k = a[\lfloor k/2 \rfloor] = b[\lceil k/2 \rceil]$.

At every step we halve the number of elements we consider, so the algorithm will terminate in $\log(2k)$ recursive calls. Assuming the comparison takes constant time, the algorithm runs in time $O(\log k)$, which is $O(\log(m+n))$, as we must have $k \leq (m+n)$ for the k^th smallest element to exist.