Fundamental Algorithms, Section 003
Homework 4, Fall 22. Due date: Wednesday, October 5, 4:55pm on Gradescope

1. Suppose the input to RSelect consists of n items all with the same key value. Suppose each pivot is chosen as the leftmost item in the subarray currently being processed. How many comparisons does RSelect perform in this case when seeking the smallest item? What about if you are seeking the largest item? Justify your answers.
2. Suppose the input is a partially sorted array $A[1:n]$ of distinct items. More specifically, the first m items are in sorted order, as is the second sequence of m items, and so on for each successive sequence of m items. More precisely, $A[rm+1] < A[rm+2] < \ldots < A[rm+m]$, for $0 \leq r \leq n/m - 1$. In addition, the only inversions are between items in adjacent sequences; so every item in the r-th sequence is smaller than every item in the $(r+2)$nd sequence; in particular i.e. $A[rm+m] < A[(r+2)m+1]$ for $0 \leq r \leq n/m - 3$.

Give an $O(n)$ time algorithm to complete the sort of array A. You may assume that n is an integer multiple of m. Justify your running time bound, and explain why your algorithm is correct.
Hint. How many items in $A[1:rm]$ might be larger than some or all of the items in $A[rm+1:rm+m]$?
You may find the following notation helpful in writing your solution: let $A_r[1:m] = A[rm+1:rm+m]$ for $0 \leq r \leq n/m - 1$.

3. Suppose you are given two sorted arrays $A[1:m]$ and $B[1:n]$ as input, where $1 \leq m \leq n$. Give an algorithm to find, for each item $A[i]$, how many items in B are smaller than $A[i]$. You may assume all the items are distinct. Your algorithm should run in time $O(m+m\log(n/m))$. You may assume that n is an integer multiple of m. Justify your running time bound and explain why your algorithm is correct.
Hint. What would you do if $m = 1$? And what would you do if $m = n$?

More generally, the term $m \log n/m$ in the target running time suggests you should spend $O(\log n/m)$ time per item in A. What can you do in this time? Then, what initial computation will allow you to get to a situation in which the additional $O(\log n/m)$ time per item suffices?

4. Suppose you are given an array $A[1:n]$ of integers in the range $[1,m]$, where $m > n$. Suppose that you are also given an uninitialized array $B[1:m]$ of integers (this means you cannot assume anything about the values in B initially). Show how to identify all duplicate items in A in time $O(n)$ (the first instance of an item in increasing i index order is a non-duplicate, the others are duplicates). You can report a duplicate item as the pair $(A[i], i)$. You will need to use array B to help.
Hint. Suppose the array B had been initialized to be all 0 prior to your use of it (so you don't have to pay for it). How would you solve the problem in time $O(n)$ in this case? Now, consider how you can get the same result without B being initialized, but with an additional $O(n)$ work.

Challenge problem. Do not submit.
Suppose you are given a set of n strings which contain a total of m characters, where $m \geq n$. Design a variant of Radix Sort to sort these strings in alphabetic order in time $O(m)$. You

may assume the alphabet size for the characters has size $O(1)$ (e.g. the English alphabet). The challenge is how to handle the fact that the strings may be of different lengths.
Comment. There is a quite different algorithm that sorts the strings in time $n + p$ where p is the sum of the lengths of the distinguishing prefixes of the set of strings; a distinguishing prefix for a string is a shortest initial sequence of characters that is unique to that string.