1. For each of the recurrence relations below, use the tree method to generate a guess $f(n)$ of the running time $T(n)$ and then prove by induction that $T(n) \in O(f(n))$.

   (a) $T(n) = 4T(n/3) + n^2$
   \[ T(1) = 1 \]

   (b) $T(n) = 4T(n/3) + n$
   \[ T(1) = 1 \]

2. Below is pseudocode for a new sorting algorithm, NewSort. You should look over the code and make sure you understand how and why the algorithm works.

   (a) Prove that NewSort is indeed a correct sorting algorithm.

   (b) How long does NewSort take to run? Prove your answer.

\[
\text{Define } \text{NewSortHelper}(A, start, end) : \\
\quad \text{if } start = end \text{ then} \\
\quad \quad \text{return} \\
\quad \text{if } start + 1 = end \text{ then} \\
\quad \quad \text{if } A[start] > A[end] \text{ then} \\
\quad \quad \quad \text{swap } A[start], A[end] \\
\quad \quad \text{return} \\
\quad t = \left\lfloor \frac{end - start + 1}{3} \right\rfloor \\
\quad \text{NewSortHelper}(A, start, end - t) \\
\quad \text{NewSortHelper}(A, start + t, end) \\
\quad \text{NewSortHelper}(A, start, end - t) \\
\text{Define } \text{NewSort}(A) : \\
\quad \text{NewSortHelper}(A, 1, A.length)
\]

3. When discussing mergesort, we gave an algorithm MERGE which took two sorted lists and returned a sorted list that contained the elements of both input lists. This algorithm took $O(n)$ time, where $n$ was the total number of elements in the two lists combined. Now consider the case of merging $k$ separate sorted lists, again with $n$ total elements in all lists combined. Find an algorithm that runs in $O(n \log k)$ time. (Hint: One option is to use a heap to help.) Show that this really is the runtime of your algorithm.