1. 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.

Define \text{NewSortHelper}(A, \text{start}, \text{end}): 
\begin{align*}
&\text{if start = end then} \\
&\quad \text{return} \\
&\text{if start+1 = end then} \\
&\quad \text{if } A[\text{start}] > A[\text{end}] \text{ then} \\
&\quad \quad \text{swap } A[\text{start}], A[\text{end}] \\
&\quad \text{return} \\
&\quad t = \lfloor \frac{\text{end}-\text{start}+1}{3} \rfloor \\
&\quad \text{NewSortHelper}(A, \text{start}, \text{end} - t) \\
&\quad \text{NewSortHelper}(A, \text{start} + t, \text{end}) \\
&\quad \text{NewSortHelper}(A, \text{start}, \text{end} - t)
\end{align*}

Define \text{NewSort}(A): 
\begin{align*}
&\text{NewSortHelper}(A, 1, \text{A.length})
\end{align*}

2. When discussing mergesort, we gave an algorithm \text{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 $O(n \lg k)$ time. (Hint: One option is to use a heap to help.) Show that this really is the runtime of your algorithm.

3. \text{SUM2}(S, t) is a a function that takes as input a set $S$ of numbers and a target number $t$. The output should be a boolean equal to ”true” if there are two (distinct) numbers in $S$ whose sum is $t$ and ”false” otherwise. Find an algorithm for \text{SUM2} and analyze it. (Any correct algorithm with a correct analysis receives some credit, but the faster it is the better.)