1 Group Sorting (66 points)

We say that an array $A$ of size $n$ is $k$-group sorted if it can be divided into $k$ consecutive groups, each of size $n/k$, such that the elements in each group are larger than the elements in earlier groups, and smaller than elements in later groups. The elements within each group need not be sorted.

For example, the following array is 4-group sorted:

\[
\begin{array}{cccccccccccc}
1 & 2 & 4 & 3 & 7 & 6 & 8 & 5 & 10 & 11 & 9 & 12 & 15 & 13 & 16 & 14
\end{array}
\]

Note that every array is 1-group-sorted, and only sorted arrays are $n$-group sorted. For the rest of this problem we will only care about deterministic algorithms (and lower bounds against deterministic algorithms). You may assume that all elements are distinct, and if you want to you may assume that $n$ and $k$ are powers of 2.

(a) Describe an algorithm that $k$-group-sorts an array in $O(n \log k)$ (i.e., in at most $O(n \log k)$ time it must turn an array which is not $k$-group sorted into one that is). Prove correctness and running time.

(b) Prove that any comparison-based $k$-group-sorting algorithm requires $\Omega(n \log k)$ comparisons in the worst case.

(c) Describe an algorithm that completely sorts an already $k$-group-sorted array in $O(n \log(n/k))$ time. Prove correctness and running time.

(d) Prove that any comparison-based algorithm to completely sort a $k$-group-sorted array requires $\Omega(n \log(n/k))$ comparisons in the worst case.

2 Sorting Different-Length Items (34 points)

We saw in class some fast sorting algorithms where we assumed that the elements were integers with the same number of digits. In this problem we change those assumptions slightly.

(a) Suppose we are given an array of integers, but instead of all integers having the same length they can each have a different number of bits. So, e.g., the number 0 or 1 takes one bit, the numbers 2, 3 take 2 bits, the numbers 4, 5, 6, 7 take three bits, etc. However, the total number of bits over all of the integers in the array is equal to $n$. Show how to sort the array in $O(n)$ time (prove correctness and running time).
(b) Suppose now that we are given an array of strings (over some finite alphabet, say the letters \texttt{a-z}). Each string can have a different number of characters, but the total number of characters in all the strings (i.e., the sum over the strings of the length of the string) is equal to \( n \). Show to sort the strings lexicographically in \( O(n) \) time. Here lexicographic order is the standard alphabetic order, so for example \texttt{a} < \texttt{ab} < \texttt{b}. 