Dear Professor,

Here's a little more detailed write-up of the algorithm I discussed with you this morning, which uses two lists to carry out the alphabet reduction used to determine an optimal Huffman encoding for some document.

We assume that we are given an alphabet consisting of individual characters and their occurrence counts across some corpus.

For convenience I refer to the heads of the alphabet list and composite-alphabet list using array notation, but we should implement this as a linked list so that shifting the head/`car` off the list is a constant-time operation.

The algorithm then works as follows:

Algorithm 1: SHIFT-AND-MERGE-HUFFMAN

Input : ALPHABET *with frequencies, unsorted*
Output: *Huffman Tree*

Triviality check: **return** ALPHABET[0] if it is the sole element

Initialize:
Sort ALPHABET with $O(n)$ or $O(n \lg n)$ algorithm
Initialize COMPOSITE as an empty doubly-linked list

while ALPHABET and COMPOSITE together have > 1 item **do**
  SMALLEST $\leftarrow$ *lesser of* ALPHABET[0] *and* COMPOSITE[0]
  Remove SMALLEST from its list
  NEXTSMALLEST $\leftarrow$ *lesser of* ALPHABET[0] *and* COMPOSITE[0]
  Remove NEXTSMALLEST from its list
  NEWNODE.$Freq \leftarrow$ SMALLEST.$Freq +$ NEXTSMALLEST.$Freq$
  Set SMALLEST and NEXTSMALLEST as children of NEWNODE
  Insert NEWNODE at the end of COMPOSITE
end
return COMPOSITE[0]

We have the option of taking an $O(n)$ step before sorting the list in order to normalize the alphabetic counts into frequencies $0 \leq freq \leq 1$. In doing so we may be able to discern enough about the data distribution to guarantee an $O(n)$ sort.

Below is an argument for why it works.

Suppose we have an input alphabet $A = \{a_0 \leq a_1 \leq \ldots \leq a_n\}$ and a composite list $B = \{b_0 \leq b_1 \leq b_2 \leq \ldots \leq b_n\}$.

Consider a point after a shift-and-merge operation has completed. The following properties should hold.

1 $b_0 \leq b_n$

This is trivially true initially (when $B = \{\emptyset\}$) and after the first shift-and-merge step (when $|B| = 1$).

Suppose elements α and β were removed from the combined composite-alphabet set $\{A \cup B\}$ and merged to form element b_0. Any subsequent shift-and-merge step would create $b_n = \gamma + \delta$. But since the shift-and-merge operation removes the smallest elements from the combined set, we know that $\alpha \leq \beta \leq \gamma \leq \delta$, therefore $b_0(= \alpha + \beta) \leq b_n(= \gamma + \delta)$.

Since all elements remaining in the combined set are no less than the ones which combined to form b_n, any further shift-and-merge operations will also result in merged values greater than b_n.

2 $b_n \leq 2b_0$.

$b_n = \alpha + \beta$ was the item added most recently to the composite list, where α and β were the smallest two items in the combined set $\{A \cup B\}$. Since α and β were both removed from the combined set, $\alpha \leq \beta = b_0 + i$ (where $0 \leq i$); so b_0 cannot have been less that α or β.

But since $\alpha \leq \beta$ and $\beta \leq b_0$, therefore $\alpha + \beta = b_n \leq 2b_0$.

3 By consequence of 1 and 2, $b_0 \leq b_n \leq 2b_0$ for any $n \geq 0$, provided B is not empty. This also requires that B will remain in sorted order throughout the operation.

4 Just to re-emphasize, $b_0 \leq b_1 \leq b_n \leq 2b_0$. So it is not possible to enter a situation in which $b_0 + b_1 \leq b_n$, even once the initial alphabet is gone and we are only carrying out the shift-and-merge operation on the composite list.