CS161 Homework 3 Problem 3-4

Algorithm

Assume the lists are sorted such that the smallest elements are at lower indices. Divide the k lists up into groups of two. To merge each group, use the same merge algorithm as used in merge sort: examine the first element in each list. Remove the smaller element and place it at the end of the new merged array. Repeat the process of comparing the first elements in each list until there are no elements left in both lists. If one list becomes empty before the other, append the rest of the elements from the non-empty list to the new list. Once the merge step is done for all $k/2$ pairs, repeat the process on the merged lists until there is only one list left, which is the final merged list.

Correctness

There are two parts to this proof: first, we must prove that all elements from all lists end up in the final list. Second, we must prove that the elements remain in sorted order.
To prove the first point, we can see from our algorithm that we traverse all lists until they are empty at each merge, so we know that at each merge, all elements get merged into the correct list. We also know that we will continue to merge lists until there is only one list left. Since no elements are lost at any given merge, that means that in the final list, all elements will be present.
To prove the second point, we can see that, for each step in the merge, we compare the first elements in each list, and put the lower one first in our merged list. We assumed that each list is sorted from lowest to highest, so that means that the first element in the list is the lowest element in the list. Therefore, when we put the lowest element into our new list, we know that it is lower than all elements that will come after it, and greater than all elements that have come before it. Therefore, each merge operation results in a sorted list, so, at the end, we still have a sorted list.

Running Time

For each merge step, we are merging k_i lists into $\frac{k_i}{2}$ lists. Let a_j be the number of elements in the jth list. We know from merge sort that the amount of time it takes to merge two lists with total m elements is $\Theta(m)$. That means that the amount of time it takes to merge two lists, a_j and a_m is $\Theta(a_j + a_m)$. Therefore, to merge all $\frac{k_i}{2}$ pairs, it takes $\sum_{j=1}^{k-1} \Theta(a_j + a_{j+1}) = \Theta(a_1 + a_2) + \Theta(a_3 + a_4) + ... + \Theta(a_{k-1} + a_k)$ to merge all pairs. Since $\sum_{j=1}^{k} j = n$, it therefore takes $\Theta(n)$ time to merge all lists at each step.
Now that we know that the merge step at each level takes $\Theta(n)$ time, we can describe the number of merge steps our algorithm performs in terms of a recurrence in terms of k. The number of merge steps to merge k lists into one list is the number of merges it takes to

merge k lists into $k/2$ lists (one merge step) plus the number of merges it takes to merge $k/2$ lists into one list. Therefore, our recurrence for the number of merge steps is of the form $T(k) = T(k/2) + \Theta(1)$. This recurrence is in the correct form for the master method, with $a = 1$, $b = 2$, and $f(k) = \Theta(1)$. We have $k^{\log_b a} = k^0 = 1$, so we are in the second case of the master method, with $f(k) = \Theta(\log^0 k) = \Theta(1)$. Therefore, $T(k) = \Theta(\log k)$. That means that to merge k lists, we will be doing $\log k$ merge steps. We know that each merge step takes $\Theta(n)$ time, so our total run time is $\Theta(n \log k)$.