CS161 Homework 3 Problem 3-4

Algorithm

Assume the lists are sorted such that the smallest elements are at lower indices. For each list, pop off the first element e, and put e and the list number j as a pair into a min-heap H with $O(logm)$ INSERT and EXTRACT_MIN (where m is the number of elements in the heap, and where the heap is ordered according to e-value only). Let A be the new, sorted array. While there are still elements the heap, extract the pair (e, j) with the minimum e from H and append e to A. Then, if list i still has elements, pop the first element off of list j and insert it into H. If list j there is no need to add another element to H.

Correctness

Since there are only a finite number of elements in the original k lists, we know that the algorithm must terminate, because we continue to put elements from each list into the heap until that list is empty, and then we continue to extract elements from the heap until the heap is empty. Since the heap contains finite elements, and each list contains finite elements, the lists and the heap will eventually become empty. By that same argument, we can say that all elements in the original k lists must end up in the final merged list, because all elements in all k lists are added into H, and all elements in H are added to A.

To prove correctness, we can use induction. Let $A[i]$ be the ith element of A. We will show that if element $A[i-1]$ is in the correct, sorted position, then $A[i]$ must also be in the correctly sorted position.

For the base case, we will show that the 0th element in A is in the correct place. To do so, we must show that the 0th element in A is the smallest element out of all elements in all k lists. Since the k lists are sorted in increasing order, the first element of each list is the smallest element in the list. Since we put all of the first elements of each list into the heap, and the heap will return the minimum of those elements, the first element in A must be the smallest.

For the inductive step, we will show that, if $A[i-1]$ is in the correct position, $A[i]$ must also be in the correct position. Let us examine the two cases for $A[i-1]$ and $A[i]$:

Case 1: $A[i-1]$ and $A[i]$ are from the same list. In this case, $A[i]$ must have been added to H after $A[i-1]$ was extracted from H. This is true because we only add a new element from a list when the old one is extracted. Since we start with at most one element from each list in H, and we don't add new elements from that list until the old one has been removed, $A[i]$ must have been added to H after $A[i-1]$ was removed. Therefore, $A[i] > A[i-1]$, because the original list was sorted, so the elements added to H later are guaranteed to be larger. In this case, the resulting list is in the correctly sorted order.

Case 2: $A[i-1]$ and $A[i]$ are from the different lists. In this case, $A[i]$ must have been in H when $A[i-1]$ was extracted from H, because we only add a new element from a list

when the old one is extracted. Since $A[i]$ and $A[i-1]$ are in different lists, it is impossible for $A[i]$ to be added after $A[i-1]$ was extracted, because the only element added after $A[i-1]$ was extracted was from the same list as $A[i-1]$. Since $A[i]$ was in the heap when $A[i-1]$ was extracted, $A[i-1] < A[i]$, because $A[i-1]$ was the minimum value in the heap at that time. In this case, therefore, the elements are in the correct order.
In both cases, we see that the elements remain in correctly sorted order as elements are appended. Therefore, the resulting list must be in sorted order.

Running Time

The size of H is at most k. Since H is a standard min-heap, both INSERT and EXTRACT_MAX run in $O(\log k)$ time. Since we insert all n elements into H, the total insertions into the heap run in $O(n \log k)$ time, and iterating through the elements runs in $O(n)$ time, so the total time to insert all elements into H is $O(n \log k)$. Similarly, since we extract all n elements from H, the total extractions from H also run in $O(n \log k)$ time, and putting the elements into the resulting array runs in $O(n)$ time, meaning the whole extraction step runs in $O(n \log k)$ time. Thus, the total run time for the algorithm is $2O(n \log k) = O(n \log k)$.