1) (10 pts) ANL (Algorithm Analysis)

Consider the problem of taking n sorted lists of n integers each, and combining those lists into a single sorted list. For ease of analysis, assume that n is a perfect power of 2. Here are two potential algorithms to solve the problem:

Algorithm A: Run the Merge Algorithm, defined between two lists, on lists 1 and 2 to create a single merged list. Then rerun the algorithm on this merged list and list 3, creating a merged list of all items from lists 1, 2 and 3. Continue in this fashion, running the Merge Algorithm n-1 times, always between the currently "growing" list and the next list to be merged into it, until list n is merged in, creating a single sorted list.

Algorithm B: Pair up the lists into $\frac{n}{2}$ pairs of lists of size n. Run the Merge Algorithm on each of these pairs. Once this phase finishes, there will be $\frac{n}{2}$ lists with 2n integers. With the new lists, repeat the process until we are left with a single sorted list.

With sufficient work and proof, determine the Big-Oh run time, in terms of n, of both of these algorithms. Clearly put a box around your final answer for both algorithms.

Analysis of Algorithm A

In this algorithm, we run the Merge Algorithm n-1 times. The first time, the list sizes are n and n. The second time the list sizes are 2n and n. In general, on the k^{th} time the list sizes are kn and n. When merging lists of these two sizes, the run-time is O(kn+n) = O(n(k+1)) ≤ cn(k+1) for some constant c. Now, we must sum up these run-times in the range of k = 1 to k = n-1. Mathematically, we have

$$\sum_{k=1}^{n-1} cn(k+1) = cn\sum_{k=1}^{n-1}(k+1) = cn\sum_{k=2}^{n} k = cn\left(\frac{n(n+1)}{2} - 1\right) = \frac{cn^3 + cn^2 - 2cn}{2} = \boldsymbol{O(n^3)}$$

Analysis for Algorithm B

In the first iteration, we run n/2 merges on lists of size n. The total run time for this iteration is $O(n^2)$, since each of the elements in all lists is involved in a single merge. (Alternatively, we have n/2 merges, each of which take O(n) time to run, for a total of $O(n^2)$.)

In the next iteration, we run n/4 merges on lists of size 2n. For the same reason as before, this step also takes $O(n^2)$ time.

The question becomes: How many iterations do we have to run before we get a single list? Notice that after the k^{th} iteration of merging lists, we have a total of $\frac{n}{2^k}$ lists left. We stop when we have a single list, so to calculate the number of iterations, set $\frac{n}{2^k} = 1$. When we solve this equation for k, we get $k = log_2 n$.

It follows that the total run time of Algorithm A is **$O(n^2 \lg n)$.**

Grading: Alg A - 1 pt for set up of summation, 3 pts for doing the summation (be lax here), 1 pt for the final Big-Oh result. Alg B - 2 pts for the initial analysis, 2 pts for calculating the number of iterations, 1 pt for the final Big-Oh result.