Runtime Proof

1 Sorting ($O(nlog_n)$)

The complexity of $O(nlog_n)$ for sorting algorithms is typically associated with **comparison-based** sorting algorithms like mergesort, quicksort, and heapsort. These algorithms sort a list by comparing elements against each other. The efficiency of the algorithm depends on the number of comparisons it makes.

1.1 Divide and Conquer

Many sorting algorithms divide the list into smaller parts, sort each part, and then combine these sorted parts. This process generally involves recursively dividing the list until you have sublists of length 1 (which are trivially sorted), and then combining these sorted lists.

1.2 $O(log_n)$ Part

Assume the list size is n, and we keep dividing the list in half, until the list reaches a list of size 1. Then we need k divisions:

$$n/2^k = 1 \tag{1}$$

Take the logarithm, we can get:

$$k = log_2 n \tag{2}$$

Recall that the base of logarithmic runtime does not matter, so we prove the $O(log_n)$ part.

1.3 $O(n)$ Part

After log_n levels of division, we end up with n sublists of size 1, which refers to the run time $O(n)$.

1.4 $O(nlog_n)$ Part

After halving the list down to single-element sublists, the merge process (which has a linear complexity in the size of the sublists) begins. This merging happens at each level of the divided list, contributing to the n part of $O(nlog_n)$.