# Solution for Merge Sort Comps
# Time Complexity (Part 2) and Proof of Correctness (Part 3)

October 20, 2025

## Part 2: Time Complexity Analysis

### Step-by-Step Cost Analysis

Let $n$ be the number of rows (people).

- **Data Load** Reading the CSV and constructing $n$ objects is $O(n)$. Computing `new_decile` $= 0.6\,P + 0.5\,J + 0.1\,C$ per row is $O(1)$, for $O(n)$ total.
- **Recursive Splitting.** Each call partitions its subarray in $O(1)$ time; there are $2n - 1$ calls overall across the recursion tree, so the aggregate splitting overhead is $O(n)$.
- **Merging.** If two sublists have combined size $k$, a single `merge` performs $k - 1$ comparisons and $k$ moves (both $O(k)$). Summed across one level of the recursion tree, this is $O(n)$ work.

### Overall Runtime Discussion

Let $T(n)$ be the worst-case running time of merge sort on $n$ elements. Then

$$T(n) = \begin{cases} \Theta(1), & n \leq 1, \\ 2T\left(\frac{n}{2}\right) + \Theta(n), & n > 1. \end{cases}$$

By the Master Theorem ($a = 2$, $b = 2$, $f(n) = \Theta(n)$, and $n^{\log_b a} = n$),

$$T(n) = \Theta(n \log n).$$

## Part 3: Proof of Correctness

### Lemma 1 (Merge Correctness)

If $L$ and $R$ are sorted by `comes_before`, then `merge(L,R)` returns a list $M$ such that

(i) $M$ contains exactly the elements of $L \uplus R$ (multiset union), and

(ii) $M$ is sorted by `comes_before`.

*Proof.*

**Property i**

The merge algorithm maintains two pointers (indices) $i$ and $j$ for lists $L$ and $R$ respectively, both initially 0. At each iteration, the algorithm compares $L[i]$ and $R[j]$ using `comes_before`, appends the smaller element to output list $M$, and advances only the corresponding pointer. This process continues until one list is exhausted, at which point all remaining elements from the other list are appended to $M$.

Since each iteration removes exactly one element from either $L$ or $R$ and appends it to $M$, and the algorithm terminates only when both lists are fully consumed (when $i = |L|$ and $j = |R|$), every element from both input lists appears exactly once in $M$. No elements are lost and none are duplicated. Therefore, $M$ contains precisely the multiset union $L \uplus R$.

**Property ii**

We prove by induction on $t$, the number of elements appended to $M$, that $M$ maintains the sorted property as a prefix invariant.

Base case ($t = 0$): Initially, $M$ is empty, which is trivially sorted.

Inductive step: Assume after $t$ steps, $M[0..t-1]$ is sorted. At step $t+1$, we append either $L[i]$ or $R[j]$ to $M$, specifically the element $e$ for which `comes_before`$(e, \cdot)$ returns true when compared with the other list's current head.

Since both $L$ and $R$ are sorted, we have:

- All elements $L[i'], i' \geq i$ in $L$ satisfy $L[i] \leq L[i']$
- All elements $R[j'], j' \geq j$ in $R$ satisfy $R[j] \leq R[j']$

The element $e$ appended at step $t+1$ is the minimum among $\{L[i], R[j]\}$ (the current heads). Since all remaining elements in both lists are $\geq$ their respective heads, $e$ is the minimum among all unprocessed elements.

By the inductive hypothesis, $M[0..t-1]$ is sorted. The last element $M[t-1]$ was chosen in a previous step as the minimum among available elements at that time. Since $e$ is now chosen as the minimum among currently available elements, and all elements previously chosen were from a set that has only grown smaller, we have $M[t-1] \leq e$. Therefore, $M[0..t]$ remains sorted.

When one list is exhausted, the remainder of the other list is appended. Since that list is already sorted and all its remaining elements are $\geq$ the last element added to $M$ by the same reasoning, sortedness is preserved.

In conclusion, by induction, $M$ is sorted throughout the merge process, and thus the final output is sorted. □

### Lemma 2 (Base Case Correctness)

If $|data| \leq 1$, then `merge_sort(data,key)` returns a sorted list containing exactly the input elements.

*Proof.* A list of size 0 or 1 is vacuously sorted and returned unchanged. □

### Lemma 3 (Merge Sort Correctness)

For any finite list $A$ of `Person` records and any valid key, `merge_sort(A,key)` returns a list that (1) is a permutation of $A$ and (2) is sorted in nonincreasing order by that key, breaking ties by name ascending.

*Proof.* By strong induction on $n = |A|$.

**Base:** For $n \leq 1$, the claim holds by Lemma 2.

**Inductive step:** Assume the claim holds for all lists of size $< n$ (with $n \geq 2$). Split $A$ into two halves $L$ and $R$ with $|L|, |R| < n$. By the inductive hypothesis, `merge_sort(L,key)` and `merge_sort(R,key)` return sorted lists that are permutations of $L$ and $R$, respectively. Applying Lemma 1 to these two sorted outputs yields a merged list $M$ that is sorted and contains exactly the multiset union of $L$ and $R$, which is precisely $A$. Thus $M$ is a sorted permutation of $A$.

Therefore, by induction, the theorem holds for all $n$. □

## Termination

The algorithm terminates because:

1. The base case ($n \leq 1$) performs no recursive calls
2. For $n \geq 2$, each recursive call operates on a strictly smaller list ($\lfloor n/2 \rfloor < n$ and $\lceil n/2 \rceil < n$)
3. The merge operation processes elements linearly and always terminates in $O(|L| + |R|)$ steps