Understanding the Recurrence Relation in Merge Sort

1 Breakdown of the Recurrence Relation

The recurrence relation for Merge Sort is:

$$T(n) = 2T\left(\frac{n}{2}\right) + \Theta(n)$$

1.1 The $2T\left(\frac{n}{2}\right)$ Term

- **Recursive Division**: This part of the recurrence represents how the array of size n is divided into two subarrays, each of size $\frac{n}{2}$.
- **"2T"**: The "2" indicates that there are two recursive calls, one for each of the two subarrays of size $\frac{n}{2}$. Each of these subarrays will be further divided in the same way, recursively.
- **Recursion Depth**: The process continues until the subarrays are reduced to size 1, at which point the recursion "bottoms out." This recursive division happens $\log n$ times, corresponding to the depth of the recursion tree.

1.2 The $\Theta(n)$ Term

- **Merge Work**: This part of the recurrence represents the work done during the merge step. After the recursive division has reached the base case and the recursion begins to unwind, the merge function combines the two sorted subarrays back into a single sorted array.
- **Work Done as the Stack Unwinds**: As the recursion unwinds, the merge work is done at each level of the recursion tree. At each level, the merge function processes all n elements, which is why this term is $\Theta(n)$.
- **Total Work**: The merge step is repeated at each level of the recursion tree, and since there are $\log n$ levels, the total work done by the merge steps across all levels is what contributes to the $\Theta(n \log n)$ overall time complexity of Merge Sort.

2 Summary

- $2T\left(\frac{n}{2}\right)$: This term describes how each recursive "divide" step further splits the array into two subarrays of size $\frac{n}{2}$. The "2" reflects that the original array is divided into two parts, and $T\left(\frac{n}{2}\right)$ reflects that each part is further processed by the recursive function.
- $\Theta(n)$: This term represents the merging work that is done as the recursion unwinds. After all the recursive divisions have been completed, the merge function combines the sorted subarrays back together, processing n elements at each level of the recursion tree.

This interpretation shows that the recurrence relation captures both the recursive division of the problem and the merging process that combines the results as the recursion unwinds. Together, they explain why Merge Sort operates with a time complexity of $\Theta(n \log n)$.