Merge Sort Time Complexity Analysis

Overview

- Merge sort has a time complexity of $\mathcal{O}(n \log n)$.
- The outer function that the user calls drives the recursion.
- The recursive part contributes the $\log n$ factor.
  - For example, if $n = 8$:
    - The first recursion breaks 8 into 2 parts of 4.
    - The next recursion breaks each 4 into 2 parts of 2.
    - The recursion bottoms out when the subarrays are of size one.
- When the stack unwinds, the merge work is done:
  - Merging the 1's into 2's.
  - Then the 2's into 4's.
  - Finally, the 4's into the final 8.
- The merge operation contributes the n factor.
- The recursive iterations are counted, representing the number of "deployments."
- For example, you don't count the individual operations in each deployment like $1 + 4 + 4 + 2 + 2 + 2 + 2 + 1 + 1 + 1 + 1 + 1 + 1 + 1 + 1$. Instead, you sum up the operations across all levels, which simplifies to $n \log n$.
- These counts are not factored into Big-O notation because they are constants.
- In Big-O notation, constants and lower-order terms are ignored, focusing on the dominant term, which is $\mathcal{O}(n \log n)$.

Recursion Tree Analysis

- Consider the recurrence relation $T(n) = 2T(n/2) + cn$.
- $T(n)$ represents the total time complexity for sorting an array of size n.
- $2T(n/2)$ represents the time complexity for the two recursive calls. Since we are dividing the array into two halves, each of size $n/2$, the total cost of these two recursive calls is $2T(n/2)$.
- cn represents the time complexity of merging the two sorted halves back together, which is $\mathcal{O}(n)$. Here, c is a constant representing the cost of merging per element.

- The recursion tree has $\log n$ levels.
  - At the root level (level 0), the work is cn.
  - At level 1, the work is cn, since it sums up to $2 \times c(n/2) = cn$.
  - This pattern continues for all levels, contributing cn work at each level.
- The total work across all levels is $cn \times \log n$, resulting in $\mathcal{O}(n \log n)$ as the overall time complexity.

Summary

- The correct interpretation of the recursion tree and the cost $T(n) = 2T(n/2)+cn$ is that the total work done across all levels sums to $\mathcal{O}(n \log n)$, where each level contributes $\mathcal{O}(n)$ work, and there are $\log n$ levels.
- The constant c represents the cost per element during the merge step, and it's factored into each level's cost but is omitted in Big-O notation.