Merge Sort Time Complexity Analysis

- Merge sort has a time complexity of $O(n \log n)$.
- 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 1+4+4+2+2+2+2+1+1+1+1+1+1+1+1 bottom, then $2 + 2 + 2 + 2$, $4 + 4$ to 8.
- 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 $O(n \log n)$.