**Overview of Merge Sort:**

Divide: Split the array into two halves.

Conquer: Recursively sort each half.

Combine: Merge the two halves to produce the sorted array.

***Time Complexity Analysis***:

1. *Dividing the Array:*
   1. At each level of recursion, the array is divided into two halves.
   2. The time complexity for splitting the array at each level is O(1), as it's just a matter of computing the midpoint and making references to subarrays.
2. *Merging the Arrays:*
   1. The merge operation takes two sorted subarrays and combines them into a single sorted array.
   2. The time complexity of merging two sorted subarrays of length n/2 each is O(n), where n is the total number of elements in the array.

***Detailed Pass Analysis:***

* *Initial Array (1st pass):*

1. Array of size n is split into two halves of size n/2 each.
2. Merging these halves back together takes O(n) time.

* *Second Level (2nd pass):*

1. Each half of size n/2 is further split into two subarrays of size n/4.
2. There are now 4 subarrays, each of size n/4.
3. Merging these subarrays back together takes 4 \* O(n/4) = O(n) time.

* *Third Level (3rd pass):*

1. Each subarray of size n/4 is split into two subarrays of size n/8.
2. There are now 8 subarrays, each of size n/8.
3. Merging these subarrays back together takes 8 \* O(n/8) = O(n) time.

* *General Level:*

1. At each level k, there are 2^k subarrays, each of size n / 2^k.
2. Merging all subarrays at this level takes 2^k \* O(n / 2^k) = O(n) time.

***Total Time Complexity:***

The total number of levels of recursion is log2(n), as the array is repeatedly divided by 2 until each subarray has only one element.

At each level, the merging process takes O(n) time.Thus, the overall time complexity is:

O(n logn)

**Summary:**

Each pass through the levels of recursion takes O(n) time for merging.

There are log2(n) levels of recursion.

Therefore, the total time complexity of merge sort is O(n \log n).