1. Explain what you think the worst-case, Big-O complexity and the best-case, Big-O complexity of merge sort is. Why do you think that?

Answer:
Merge Sort (Merge Sort) is a very efficient, stable, comparison-based sorting algorithm, its basic idea is to merge two or more ordered tables into a new ordered table. The time complexity of merge sort is not affected by the input data, and the time complexity is the same whether the data is partially sorted or completely unordered.

Both the worst-case and best-case time complexity of merge sort is O(n log n), where n is the number of elements to be sorted. This is because merge sort always splits the array in half, sort them recursively, and then merge the results. This recursive and merging process takes time proportional to the logarithm of n, so the total time complexity is O(n log n).

Why would I think that? First, the recursive nature of merge sort means that it continuously breaks the problem down into smaller parts until each part contains only one element. This decomposition process requires log n layers (because it cuts the problem size in half each time), and each layer needs to process n elements. So, the total time complexity is O(n log n).

Second, the merge operation of merge sort is also linear, that is, the time to merge two ordered arrays is proportional to the total number of elements in the two arrays. Therefore, at each level of recursion, the time complexity of the merge operation is also O(n). Since there are log n layers, the total time complexity is still O(n log n).

Finally, it should be noted that the spatial complexity of merge sort is also O(n), because it requires additional space to store the ordered array produced during the recursive process. This is a disadvantage of merge sort over other in-place sort algorithms, such as quicksort. However, in some cases, if the memory space is sufficient and stability is required, merge sort is still a good choice.

2. Explain what you think the worst-case, Big-O complexity and the best-case, Big-O complexity is for this iterative merge sort. Why do you think that?

Answer:
Iterative merge sort is more complex to implement than recursive merge sort because it requires explicit management of the partitioning, merging, and intermediate state of storage of the data. However, from a time complexity perspective, both the worst-case and best-case time complexity of iterative merge sort is still O(n log n), where n is the number of elements to be sorted.

Why do you think so? First, the time complexity of the basic operation of merge sort - merging two ordered arrays - is linear, i.e. O(n). In an iterative implementation, this does not change. The main difference is how you manage the partitioning and merging of the data.

In iterative merge sort, we can use a strategy similar to divide-and-conquer, but through iteration instead of recursion. We can treat the original array as a series of sorted subarrays of length 1, which are then iteratively merged into longer sorted subarrays until the entire array is sorted.

At each iteration, we select adjacent sorted subarrays to merge. Since the time complexity of each merge operation is linear, and we need about log n iterations to sort the entire array (since each iteration roughly doubles the length of the subarray), the total time complexity is O(n log n).

This time complexity is the same in both the worst-case and best-case scenarios because we need to perform the same number of merge operations regardless of the initial order of the input data. Each merge operation needs to process the same number of elements, and the number of merges is determined by the length of the array and the iteration strategy, regardless of the specific order of the input data.

Therefore, the time complexity of merge sort is O(n log n) for both recursive and iterative implementations, and this complexity is the same in both worst-case and best-case scenarios. This is because the basic operation of merge sort and the nature of the divide-and-conquer strategy determine the upper and lower limits of its time complexity.