1

In worst case, the big-O complexity of merge sort is O(nlog(n)). It divides the array into two temporary arrays and repeats this process until all temporary arrays only contain one item. This dividing process takes log(n) steps. When it merges two temporary arrays in the same level, it needs n steps to compare the items in these two arrays and put them in sorted order. It has log(n) levels, with each level taking n steps. So the worst-case complexity is O(nlog(n)).

In best case, the big-O complexity of is O(nlog(n)). Even if the array is sorted, the dividing process is as same as the worst case and it takes log(n) steps. When it merges two temporary arrays originally in sorted order, it needs less running time, roughly n/2 steps. Therefore, the complexity is also O(log(n)).

2

In worst case, the big-O complexity of iterative merge sort is O(nlog(n)). It is obvious that the running time of the outer for loop is log(n) because index m doubles in each iteration until it gets to the size of array. And the inner for loop runs n/m times, in each iteration the running time of merge() is 2m. Therefore, the inner loop takes n steps and the O complexity is O(nlog(n)).

In best case, the big-O complexity of iterative merge sort is O(nlog(n)). Similarly, the running time of the outer for loop is log(n). And the running time of the inner for loop is n/2m. If the array is sorted, the running time of merge() is m because it only needs to compare all the items in one array with those in the other array. So the inner loop takes n/2 steps and the complexity is still O(nlog(n)).