2)

Our function uses a divide step where the array is divided into halves recursively until each small-array has just one singular element. This is of O(logn) time. Then there is a merge step, where all the small-arrays are merged. In the event of worst case, it takes O(n) time to merge the small-arrays of size n/2 each. Since the levels of recursion are O(logn), the total complexity of this step is O(nlogn).

3)

Original given array: [8, 42, 25, 3, 3, 2, 27, 3]

- Split into two small-arrays: [8, 42, 25, 3] and [3, 2, 27, 3]
- Split into two small-arrays: [8, 42] and [25, 3], [3, 2] and [27, 3]
- Merge small-arrays: [8, 42] and [3, 25], [2, 3] and [3, 27]
- Merge arrays: [3, 8, 25, 42] and [2, 3, 3, 27]
- Merge arrays: [2, 3, 3, 8, 25, 27, 42, 3]
- Final merge/sort: [2, 3, 3, 3, 8, 25, 27, 42]

As a result, the array is now sorted.

4)

The number of steps is consistent with the complexity analysis. In the worst case scenario, the number of steps is of nature nlogn. The merge sort algorithm divides the array into smaller halves and merges them back together, thus the expected time complexity comes to O(nlogn).