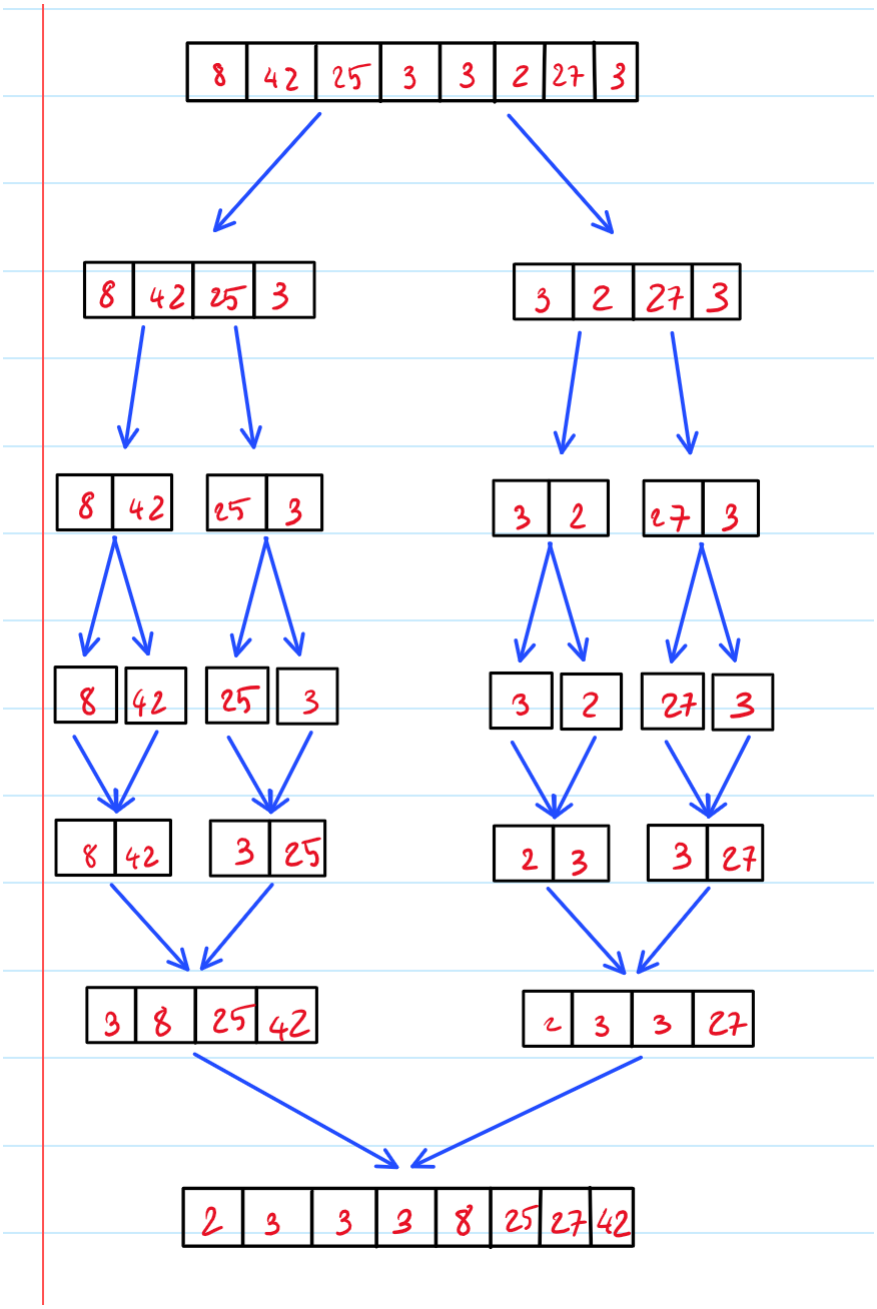
EXERCISE #1

2. The merge_sort function in the code recursively divides the array into two halves until it reaches arrays of size 1. This process forms a binary tree and takes log n levels to reach the base case, where n is the size of the array. The merge function in the code merges two sorted halves back into a single sorted array. Since the merge function is called once at each level of the binary tree formed by the merge_sort function, and there are log n levels, the total time complexity of the algorithm is O(nlogn). This is because for each of the log n levels, we do n work to merge all elements at that level.

3.



the merge function in the code is responsible for merging two sorted arrays back together by iterating over the elements of the two arrays, comparing the current elements, and adding the smaller one to the merged array. This process continues until all elements from both arrays have been added to the merged array. The merge_sort function controls the recursive division of the array and the merging of the halves.

4.

Yes, the number of steps is consistent with the complexity analysis of O(n log n)

The merge_sort function as discussed above is a logn function

The merge function is a n function

So, for each of the log n levels, we do n work to merge all elements at that level. Which has a complexity of O(nlogn) which is consistent with the analysis