ex1.pdf Group 17 Lab 3

2. Argue that the overall algorithm has a worst-case complexity of O(nlogn). Note, your description must specifically refer to the code you wrote, i.e., not just generically talk about mergesort.

The function splits the array into two halves (merge_sort(arr, low, mid) and merge_sort(arr, mid+1, high)). This gives us:

$$T(n) = 2T(n/2) + O(n)$$

where O(n) comes from merging step.

When merging, it takes O(n) time to merge two sorted halves of size n/2 each. The while loops ensure that all elements are copied, also O(n).

Using the recurrence relations:

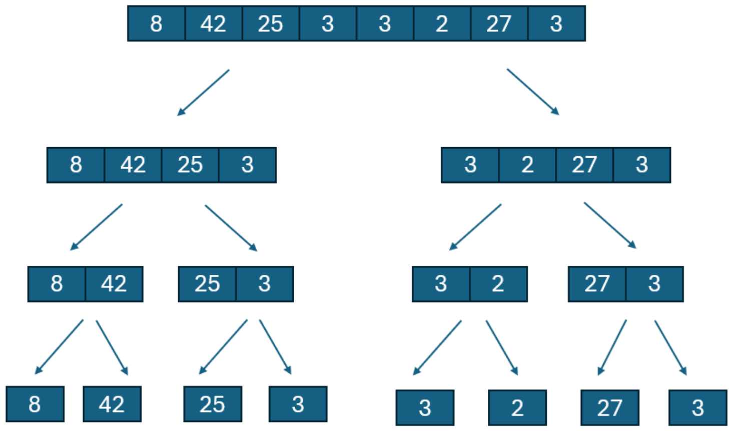
$$T(n) = 2T(n/2) + O(n)$$

We then get:

$$T(n) = O(n\log n)$$

Thus, the overall **worst-case time complexity of the merge sort algorithm (as implemented above) is O(nlogn).**

3. Manually apply your algorithm to the input below, showing each step (similar to the example seen in class) until the algorithm completes and the vector is fully sorted. Explanation should include both visuals (vector at each step) and discussion.



1. Raw data set

2. Break data into 2 halves

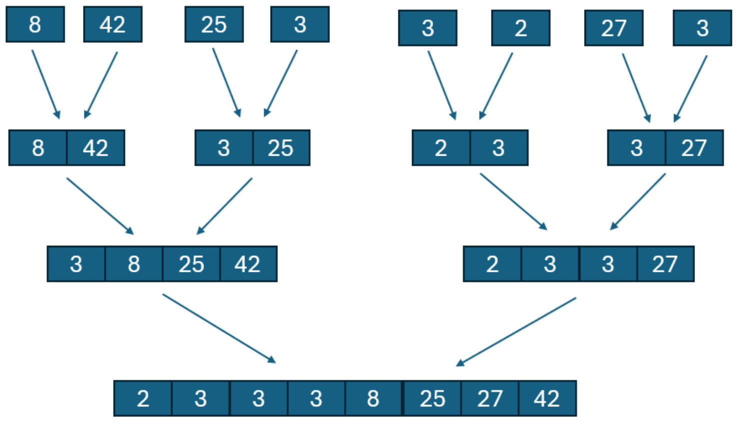
3. Break the 2 halves into 2 halves

4. Break these halves into smaller ones

1. Dataset is taken
2. The data is broken down into 2 halves
3. Using the recursion, these 2 halves are then broken down into another 2 halves – making it 4 small sets of data
4. Until it is broke down into another halves – this makes the data a single element

After all the data are broken down into smaller pieces, then merge begins to happen as follows.

1. Since single element ius considered sorted this is where the first step happens
2. Then next is it will merge the single elements into two in sorted order
3. Repeating this process having 2 sorted elements into 4 sorted elements
4. Finally, the 2 halves are now merged together with all the elements being sorted in order



1. Single elements are considered sorted

2. Merge back in sorted order

3. Merge back larger halves in sorted order

4. Final merge

4. Is the number of steps consistent with your complexity analysis? Justify your answer.

The array is divided into half in each step, giving **log(n)** levels of recursion where n = 8, therefore log (8) = 3.

Each merge operation takes O(n) time at each level. Therefore, the total complexity is **O(nlogn).**

$$O\ (n \log n) = O\ (8 \log 8) = O\ (8 \times 3) = O(24)$$

Hence, the number of steps observed in the manual process aligns with the theoretical complexity of **O(nlogn).**