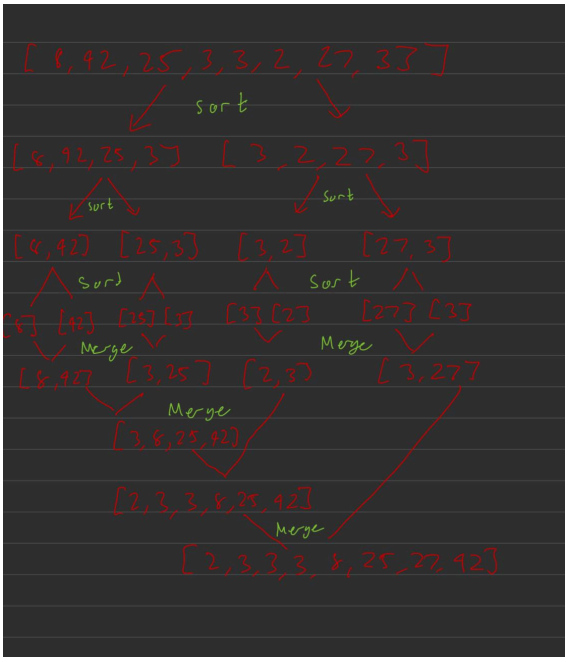
Exercise 1

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. [0.4 pts]

Based on my code, the algorithm in fact has a worst case complexity of O(nlogn). The algorithm uses a divide-and-conquer strategy, recursively dividing the array into halves until reaching single-element subarrays. This makes the depth of this recursive divide-and-conquer algorithm log_2n where n is the number of elements. The merging operation, conducted by the merge(arr, low, mid, high) function, has a time complexity of O(n). Due to the fact that the merge function happens at every level of recursion, there are log_2n levels, making the overall complexity O(nlogn). The recursion tree has a height of log_2n where n is also the number of elements. Combining the division and merging steps, the overall worst-case time complexity is O(nlogn), aligning with the observed behavior of the algorithm.

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 [0.4 pts]



A visualization of what happens within the sort/merge functions. Initially the array keeps breaking down by halves until it gets to an array of length 1, and then the merge function creates temporary arrays that keeps regrouping the split arrays in order until it gets the desired final array which is sorted in ascending order

4. Is the number of steps consistent with your complexity analysis? Justify your answer. [0.2 pts]

Yes, the number of steps is consistent with my complexity analysis of O(nlogn). The reason being due to the fact that each level of recursion has a linear amount of work being done at the merging function, therefore the overall number of steps is proportional to nlogn,