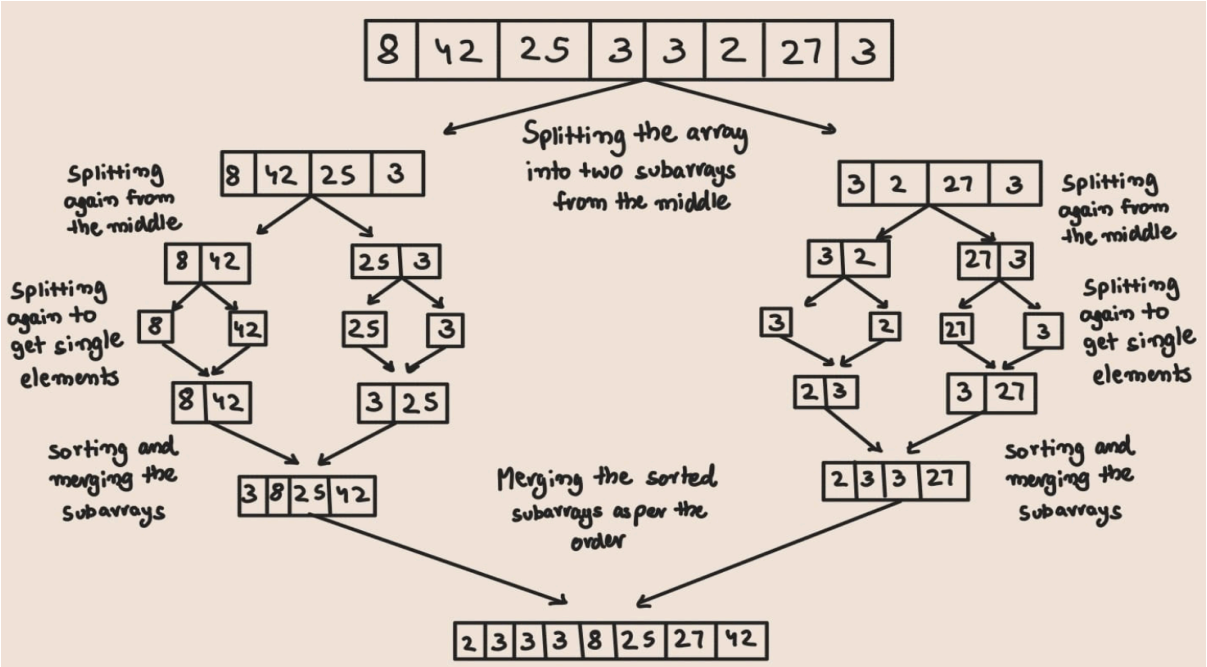
2. Argue that the overall algorithm has a worst - case complexity of O(nlogn).

O(n) is the worst-case complexity for the merge() function because the while loop in this function iterates n number of times which is the input size to combine two sorted sub arrays. Whereas, O(logn) is the worst-case complexity for the merge_sort() function because it divides the array into half and sorts it using recursion. Hence, the overall algorithm has a worst-case complexity of O(nlogn) because it is the product of worst-case complexity for the process of splitting, sorting, and merging.

3. Manually apply your algorithm to the input below, showing each step until the algorithm completes and the vector is fully sorted.



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

Yes, the number of steps are consistent with our complexity analysis. When we use Merge Sort on an array of 8 elements, it keeps splitting the array in half, $\log_2 8 = 3$ times. Then, at each of these split levels, it merges all the subarrays back together. Since there are 8 elements in the given array, there will be 8 merging steps in total. This fits with our analysis of the O(nlogn) complexity. As $n = 8$ and $\log_2 8 = 3$, there will be a total of $8 \times 3 = 24$ steps. Therefore, the steps the algorithm takes do match up with the theory that its complexity is O(nlogn).