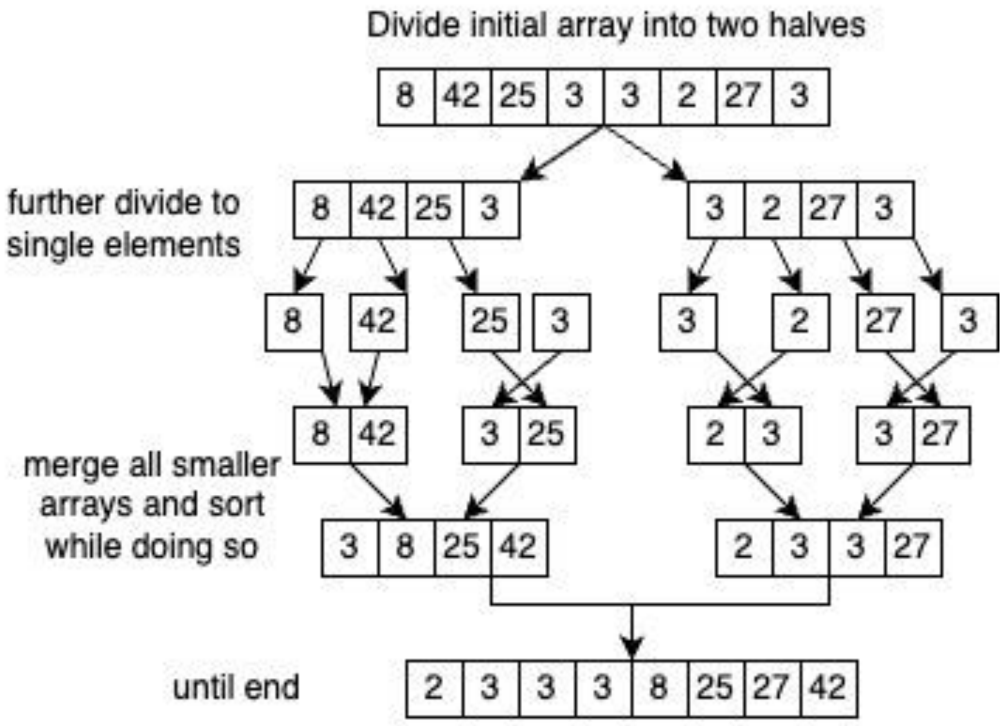
Questions for Exercise 1: ENSF 338 Lab 3

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

*The worst-case time complexity of merge() is O(n) because it processes each element once while merging. The merge_sort() function recursively splits the array into two halves, resulting in O(log n) levels of recursion. Inside the merge_sort() function there is a call to merge() after the each recursive division so that in the end, each recursive call requires O(n) operations for merging. Thus, the overall time complexity of the algorithm is O(n log n) since it is essentially O(n) * O (nlogn) because of each recursive level's call to merge().*

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


Divide initial array into two halves
8 42 25 3 3 2 27 3
further divide to single elements
8 42 25 3
3 2 27 3
8
42
25
3
3
2
27
3
8 42
3 25
2 3
3 27
merge all smaller arrays and sort while doing so
3 8 25 42
2 3 3 27
until end
2 3 3 3 8 25 27 42

- ***Initial Division:*** *The array is recursively divided into smaller subarrays until each subarray contains a single element.*

- ***Merging Process**: The single-element subarrays are then merged back together in sorted order, sorting for each level of recursive call that was made. This merging process continues until the entire array is merged back into a single sorted array.*

Question 4: Is the number of steps consistent with complexity analysis earlier?

Yes, for the array [8, 42, 25, 3, 3, 2, 27, 3] with n = 8, the division steps are O(log n) = $\log_2 8$ = 3, and at each level of recursion, merging takes O(n) = 8 work. Therefore, the total work is O(8 × 3) = O(24), which aligns with the O(n log n) complexity.