1.2. Argue that the over all 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 merge sort function splits the array into two halves at each recursive call, reducing the size of the problem.
- This splitting continues until each subarray has one element, creating about log(n) levels of recursion.
- At each recursion level, the merge function processes all the elements in the current section of the array, taking O(n) time.
- Since merging happens at every level, the total time is the product of the levels and the work per level: O(n) * O(log n) = O(n log n).

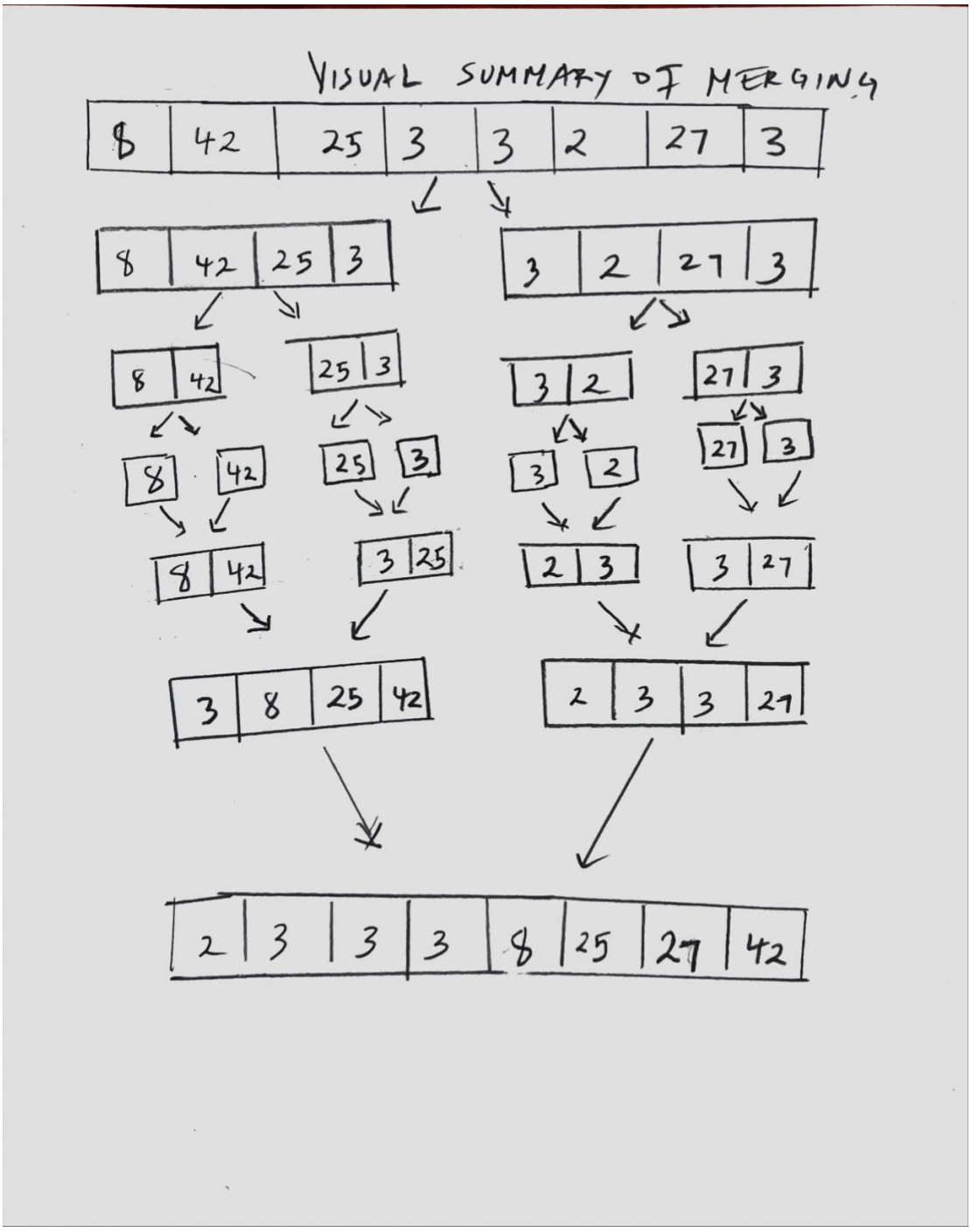
1.3. Manually apply your algorithm to the input below, showing each step (like the example seen in class) until the algorithm completes and the vector is fully sorted. Explanation should include both visuals (vector at each step) and discussion.

- the algorithm follows these steps:
- **Divide the Array:**
  The array is split into two halves:
  - Left half: [8, 42, 25, 3]
  - Right half: [3, 2, 27, 3]
- **Sort the Left Half [8, 42, 25, 3]:**
  - Split it into [8, 42] and [25, 3].
  - [8, 42] is divided further into [8] and [42] (which are already sorted) and merged back to [8, 42].
  - [25, 3] is split into [25] and [3] and merged to form [3, 25].

- Finally, merge [8, 42] with [3, 25]:
  - Compare 8 with 3, pick 3.
  - Then, pick 8, followed by 25 and 42.
  - This gives the sorted left half: [3, 8, 25, 42].

- **Sort the Right Half [3, 2, 27, 3]:**
  - Split it into [3, 2] and [27, 3].
  - [3, 2] becomes [3] and [2], and merging them produces [2, 3].
  - [27, 3] becomes [27] and [3], and merging them produces [3, 27].
  - Merge [2, 3] and [3, 27]:
    - Compare 2 with 3, pick 2.
    - Then, pick the 3's and finally 27.
    - This results in the sorted right half: [2, 3, 3, 27].
- **Final Merge:**
  Merge the two sorted halves [3, 8, 25, 42] and [2, 3, 3, 27]:
  - Compare the first elements: 3 and 2. Pick 2.
  - Continue comparing and merging step-by-step:
    - Pick 3 from the left, then the 3's from the right, followed by 8, 25, 27, and finally 42.
  - The fully merged and sorted array is: [2, 3, 3, 3, 8, 25, 27, 42]

Visual summary


VISUAL SUMMARY OF MERGING
8 42 25 3 3 2 27 3
8 42 25 3
3 2 27 3
8 42
25 3
3 2
27 3
8
42
25
3
3
2
27
3
8 42
3 25
2 3
3 27
3 8 25 42
2 3 3 27
2 3 3 3 8 25 27 42

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

Yes, The steps are consistent with an O(n log n) complexity because the algorithm divides the array into halves, creating about $\log_2(n)$ levels, and then processes every element at each level during the merge. For example, with 8 elements, there are roughly 3 levels of recursion, and each level involves merging all 8 elements, which directly aligns with the n·log n growth.