Exercise #1 – Merge [1 pt]

1)Implement a basic version of that function, using the code seen in class for the rest of the algorithm

Code provided on files.

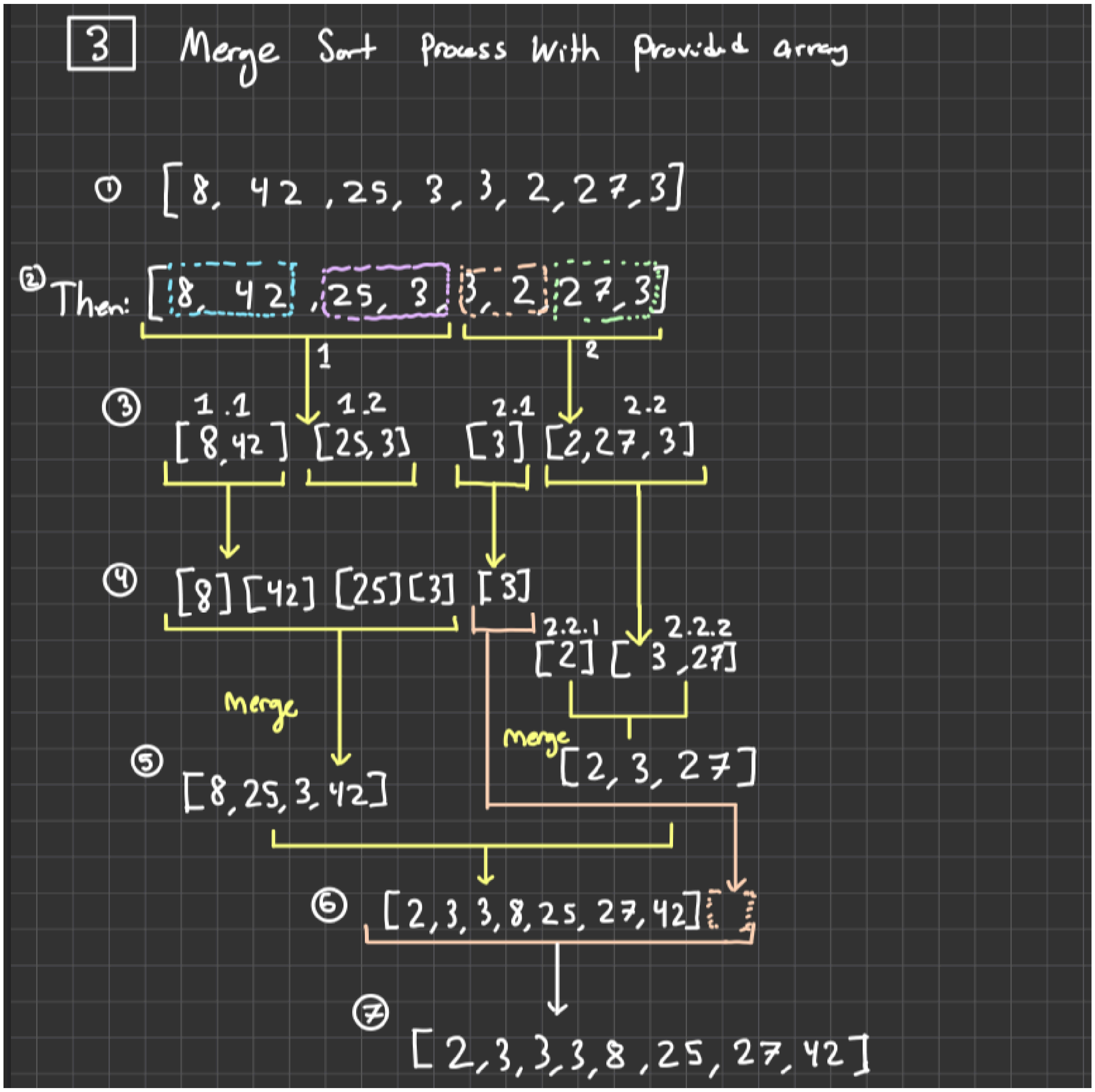
2)Argue that the overall algorithm has a worst-case complexity of O(nlogn). Note, your description must specifically refer to the code you wrote, i.e., not just generically talk about mergesort.

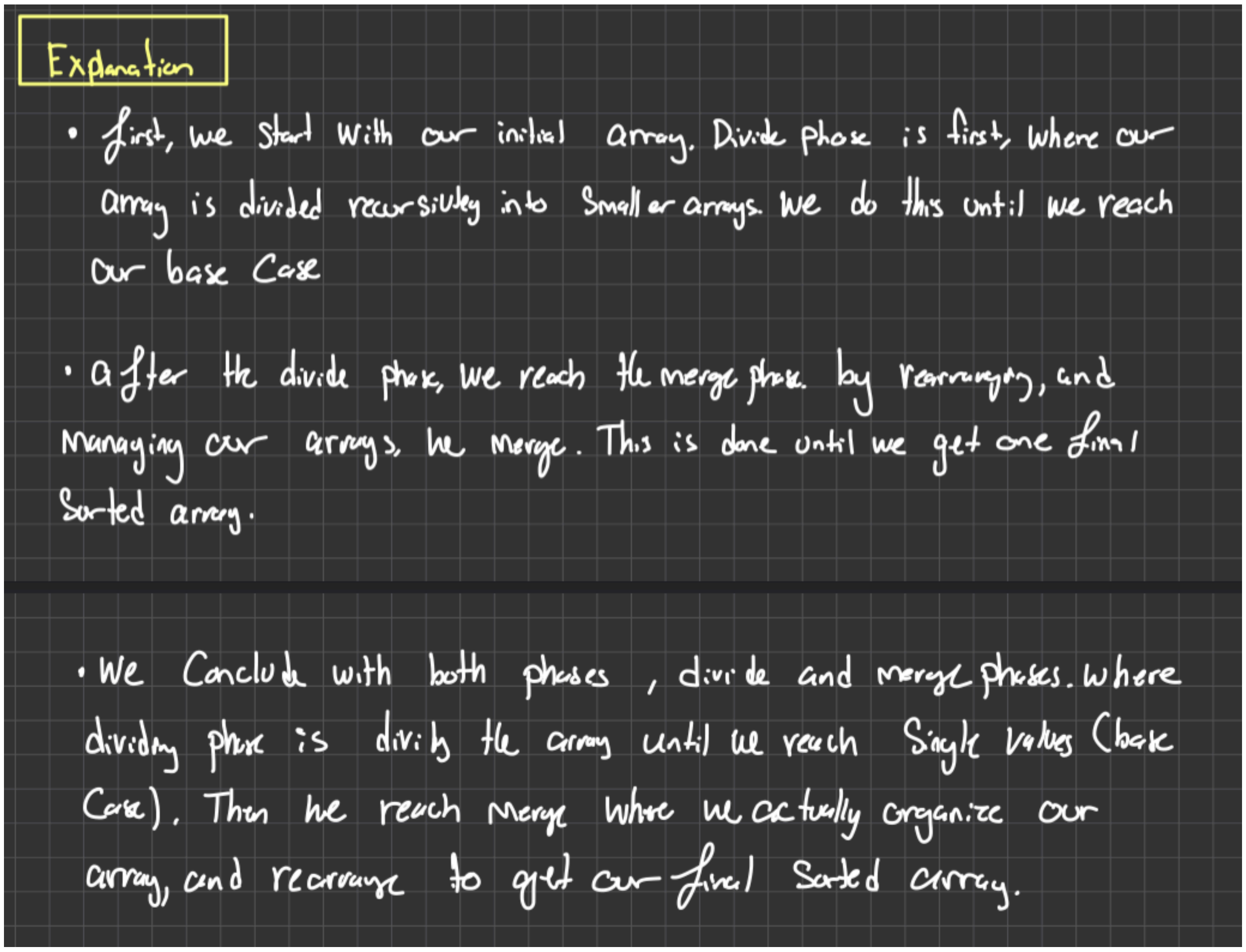
The merge sort code implemented follows the divide and conquer process. For our divide phase , we use the “merge_sort” function, which divides our array by half until we reach our base case , two base values. The division follows “merge_sort(arr, low, mid)” and “merge_sort(arr, mid+1, high)”. Each recursive iteration has a logarithmic complexity, which follows O(logn)

Secondly , in the merge phase we implemented the merge function, where this merges two arrays into one single. By comparing and rearrangement using the while loops in our code, we finish this process. This process follows a linear approach, denoted O(n), where n is the number or arrays merged.

After multiplying both complexities (divide and merge phases) we get O(log n) and O(n) which gives us the promised complexity of O(n log n).

Exercise #1 /2


3
Merge Sort Process With Provided array
① [8, 42, 25, 3, 3, 2, 27, 3]
② Then: [8, 42, 25, 3, 3, 2, 27, 3]
1
2
③ 1.1 1.2 2.1 2.2
[8, 42] [25, 3] [3] [2, 27, 3]
④ [8] [42] [25] [3] [3]
2.2.1 2.2.2
[2] [3, 27]
Merge
Merge
[2, 3, 27]
⑤ [8, 25, 3, 42]
⑥ [2, 3, 3, 8, 25, 27, 42]
⑦ [2, 3, 3, 3, 8, 25, 27, 42]



Explanation

- first, we start with our initial array. Divide phase is first, where our array is divided recursivley into smaller arrays. We do this until we reach our base case

- after the divide phase, we reach the merge phase. by rearranging, and managing our arrays, we merge. This is done until we get one final sorted array.

- We conclude with both phases, divide and merge phases. where dividing phase is divides the array until we reach single values (base case). Then we reach merge where we actually organize our array, and rearrange to get our final sorted array.

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 the complexity analysis.

In our divide phase our merge_sort function divides the arrays into halves until our case is reached. We make recursive calls for each division of the array. The number that is provided in our recursive tree follows a logarithmic relation due to the division by 2. This demonstrates O(log n) complexity.

In the second phase (merge) out merge function merges our two assorted arrays, and rearranges the elements. This follows a linear approach , following a complexity O(n).

Thus providing our promised overall complexity of O(n log n) after combining both processes , making the number of steps consistent with our complexity.