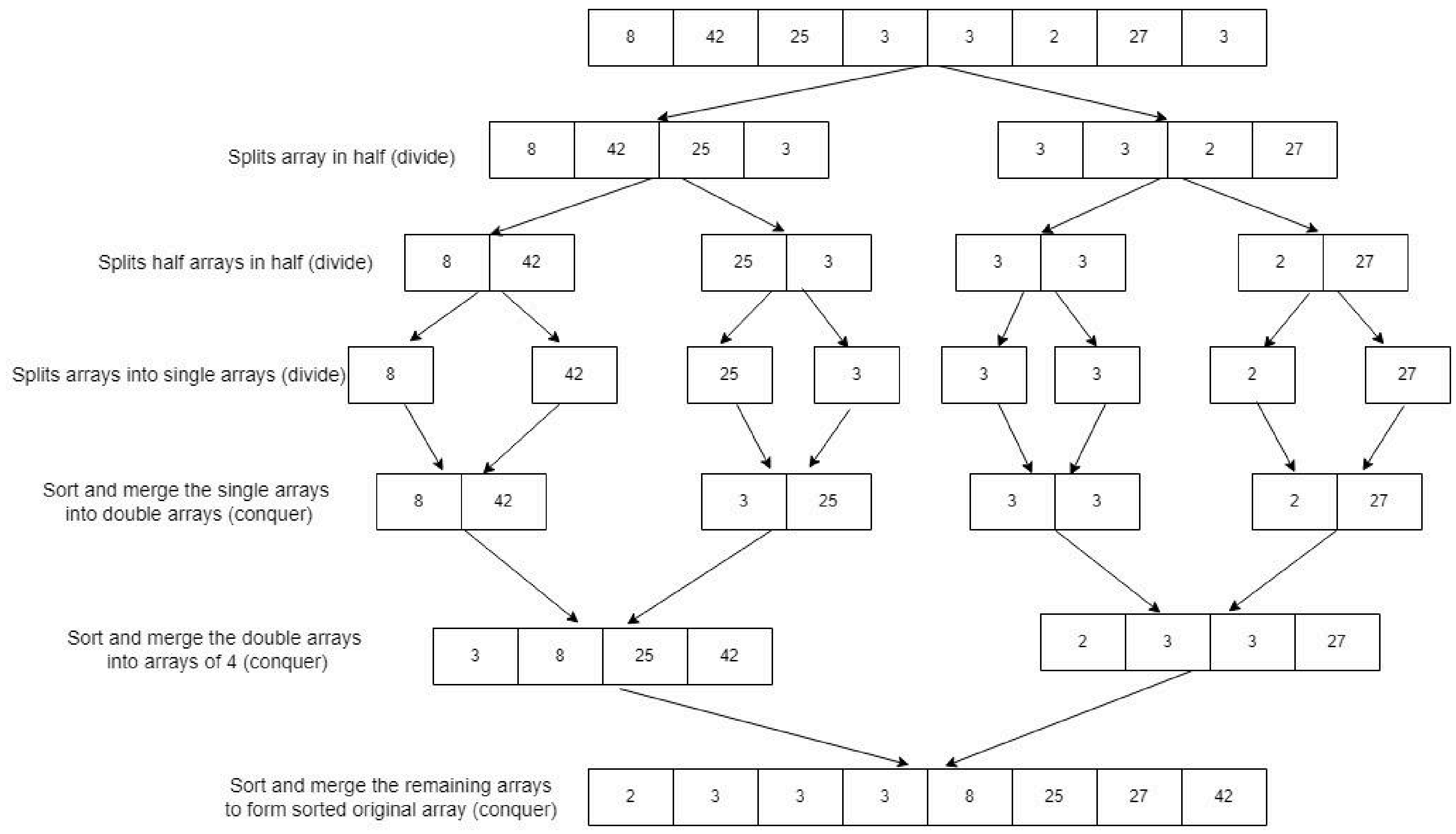
2. The merge_sort function continues to recursively call itself, dividing into 2 and leading to a tree with complexity O(log n) always since it is splitting itself in half with each call. The merge function traverses the two sorted sub-arrays produced by the merge_sort function in a linear fashion where it goes through each of the elements of the sub-arrays once, adding the smallest one into the merged array first. Due to the function going through each element of the total list only once and the time it takes is proportional to the size of the array, it has a complexity of O(n). Thus, for the entire sort algorithm, the worst-case complexity is O(nlogn).

3.


8 42 25 3 3 2 27 3
Splits array in half (divide)
8 42 25 3
3 3 2 27
Splits half arrays in half (divide)
8 42
25 3
3 3
2 27
Splits arrays into single arrays (divide)
8
42
25
3
3
3
2
27
Sort and merge the single arrays into double arrays (conquer)
8 42
3 25
3 3
2 27
Sort and merge the double arrays into arrays of 4 (conquer)
3 8 25 42
2 3 3 27
Sort and merge the remaining arrays to form sorted original array (conquer)
2 3 3 3 8 25 27 42

4. Following the merge sort complexity of O(nlog(n)), we can estimate that the algorithm should have around 24 steps for an array of size n = 8. The actual number of steps is 27 which is close the the approximated value. The difference is likely due to the complexity representing the asymptote rather than the exact number of steps, even though the complexity is always O(nlog(n)) no matter the input case.