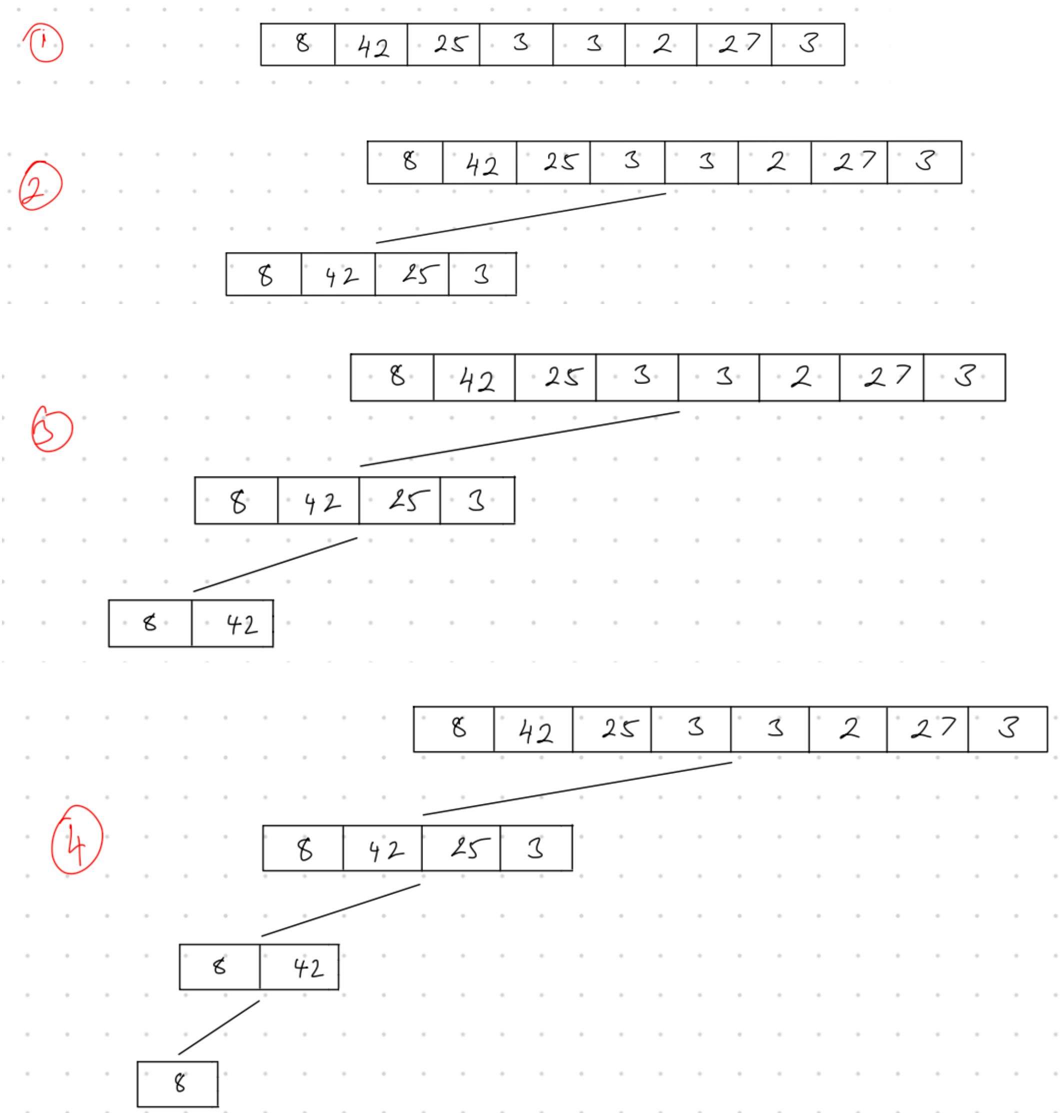
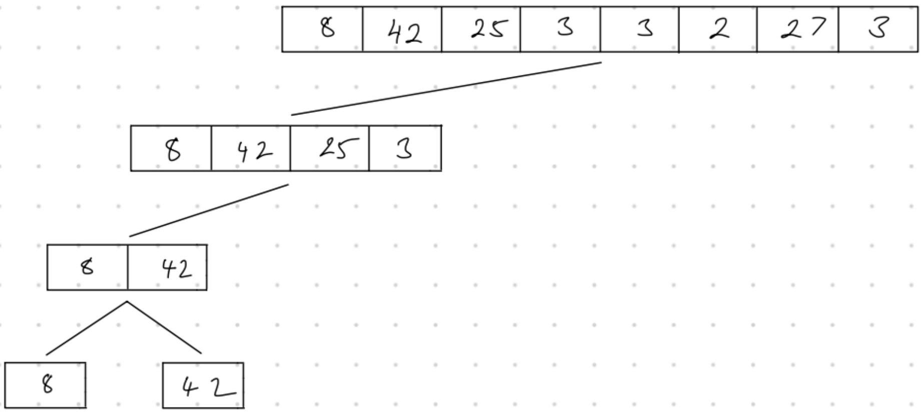
2. The overall algorithm has a worst-case complexity of *O(n*log(n))* due to the two functions involved in the algorithm, merge_sort and merge. The merge function merges two sorted sub-arrays together. To do this, it iterates through both sub-arrays simultaneously and then compares the elements to merge them. Since the sub-arrays are already sorted from the merge_sort function, the complexity of the merge function is linear with respect to the size, *n*, of the sub-arrays that are being merged; so, *O(n)*. The merge_sort function recursively divides the arrays into sub-arrays until it reaches sub-arrays of size 1. The number of recursive calls depends on the size of the array, *n*, and since each recursive call splits the array by half, there will be $log_2(n)$ levels in the recursive calls tree. So, the algorithm altogether has a worst-case complexity of $O(n * log_2(n))$.

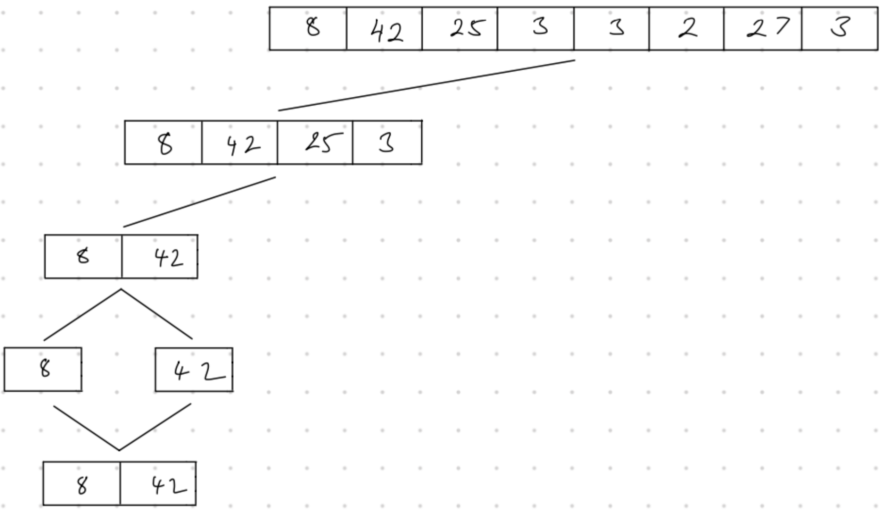
3. Manually applying the merge sort algorithm to the given vector:



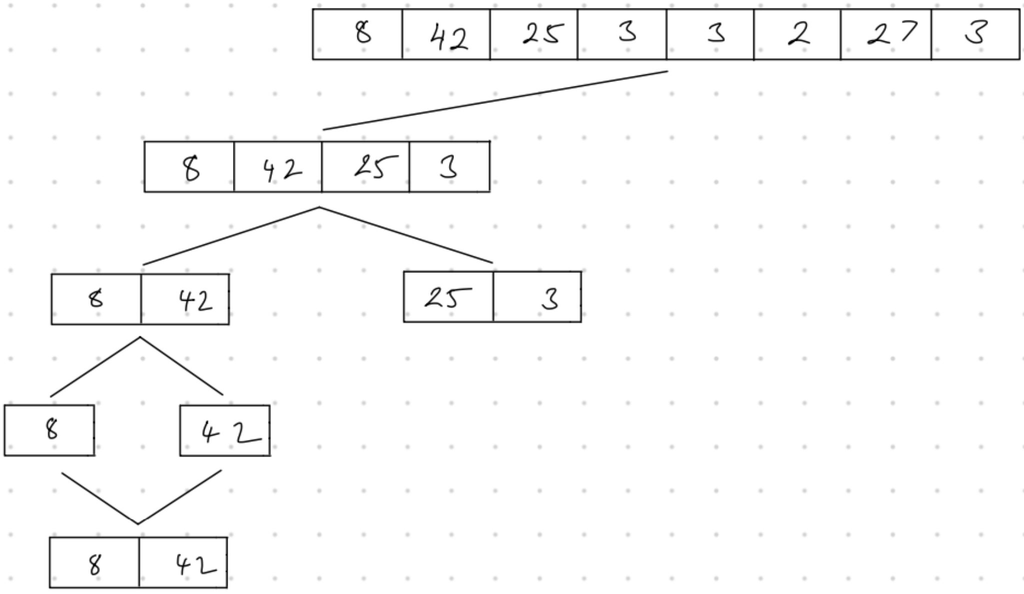
5


8 42 25 3 3 2 27 3
8 42 25 3
8 42
8
42

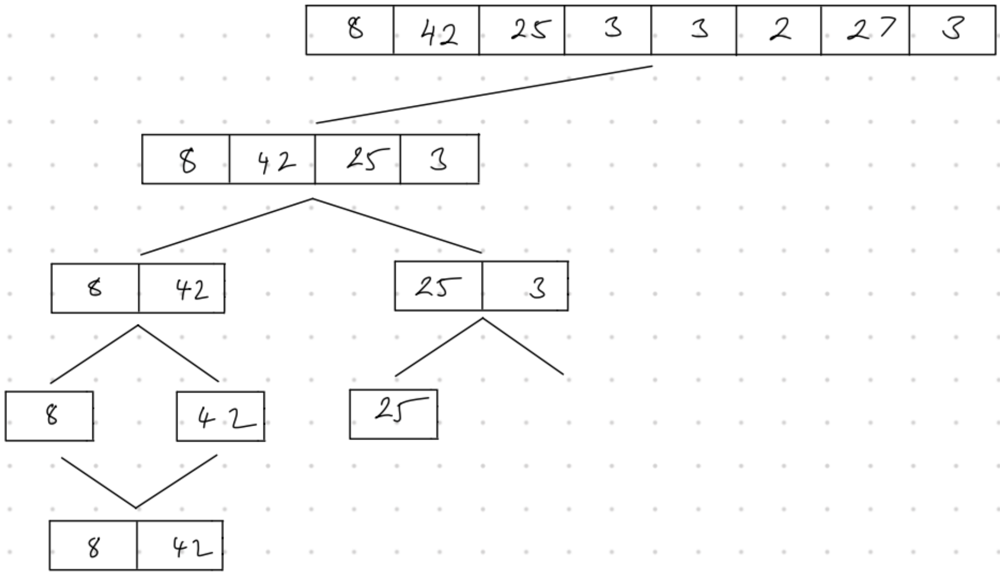
6


8 42 25 3 3 2 27 3
8 42 25 3
8 42
8
42
8 42

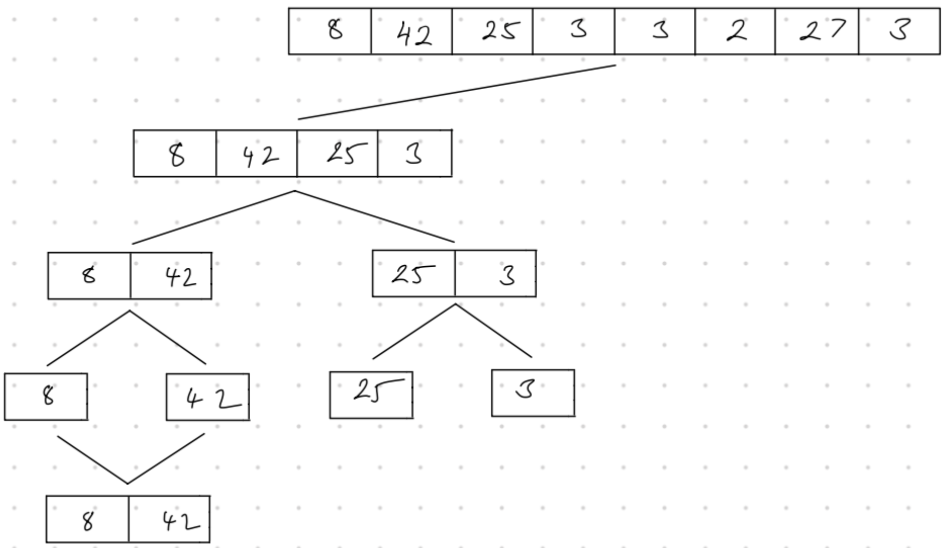
7


8 42 25 3 3 2 27 3
8 42 25 3
8 42
25 3
8
42
8 42

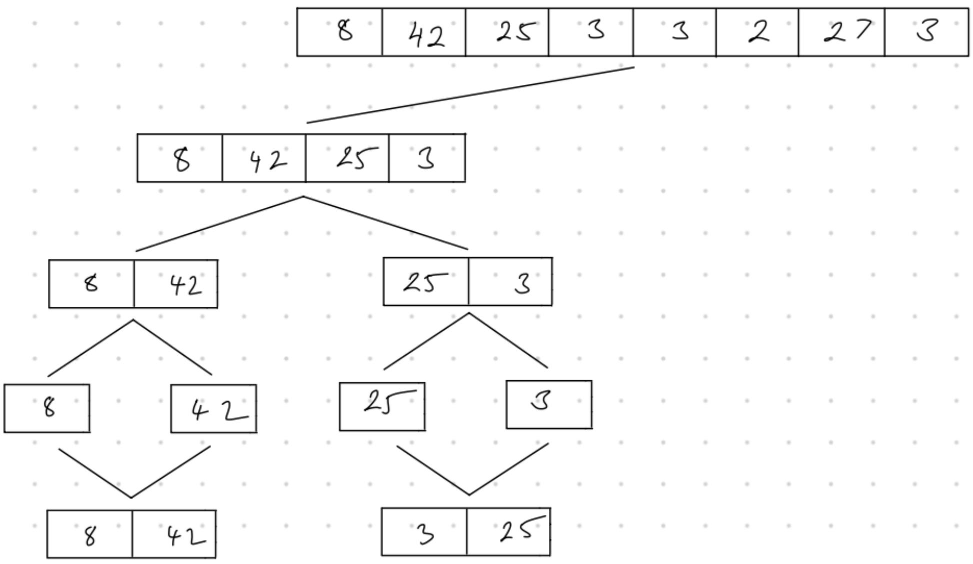
8


8 42 25 3 3 2 27 3
8 42 25 3
8 42
25 3
8
42
25
8 42

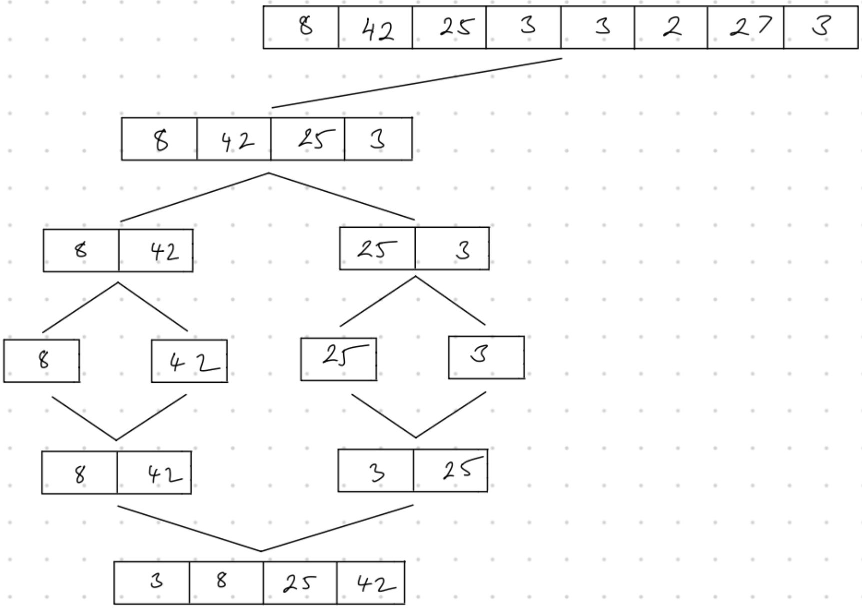
9


8 42 25 3 3 2 27 3
8 42 25 3
8 42
25 3
8
42
25
3
8 42

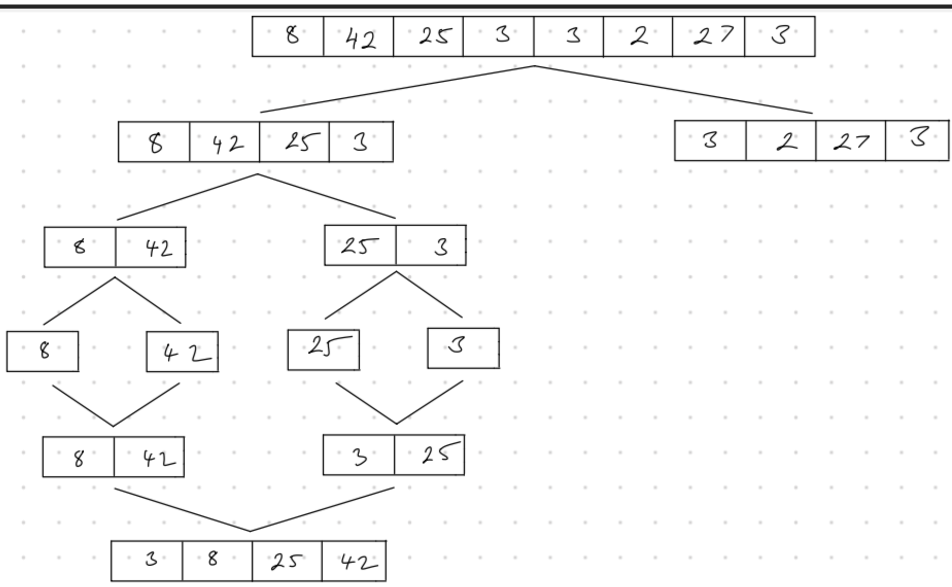
10


8 42 25 3 3 2 27 3
8 42 25 3
8 42
25 3
8
42
25
3
8 42
3 25

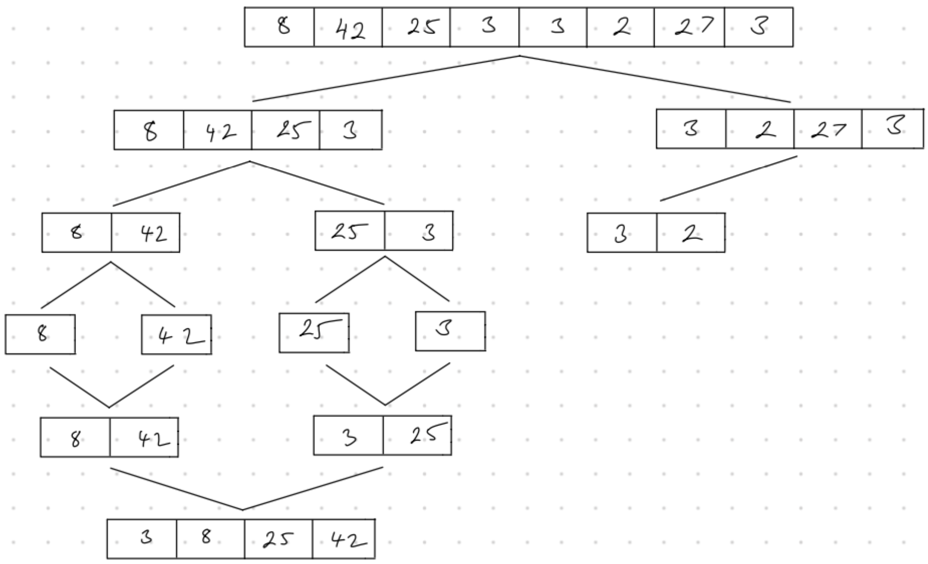
11


8 42 25 3 3 2 27 3
8 42 25 3
8 42
25 3
8
42
25
3
8 42
3 25
3 8 25 42

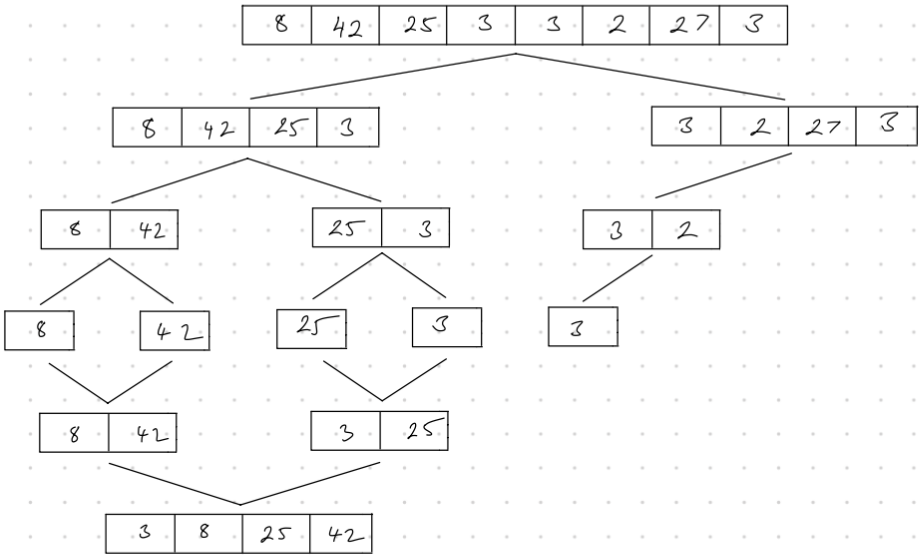
12


8 42 25 3 3 2 27 3
8 42 25 3
3 2 27 3
8 42
25 3
8
42
25
3
8 42
3 25
3 8 25 42

13

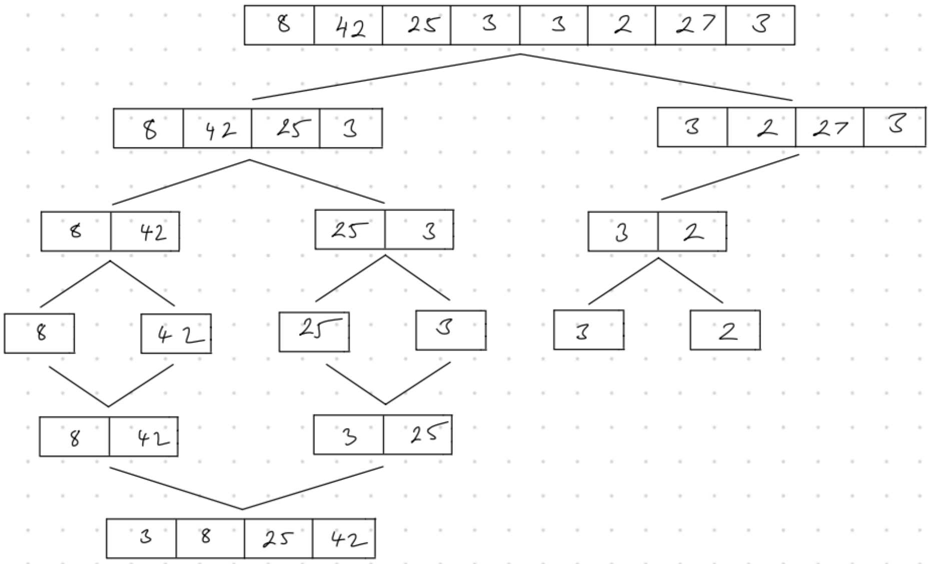

8 42 25 3 3 2 27 3
8 42 25 3
3 2 27 3
8 42
25 3
3 2
8
42
25
3
8 42
3 25
3 8 25 42

14



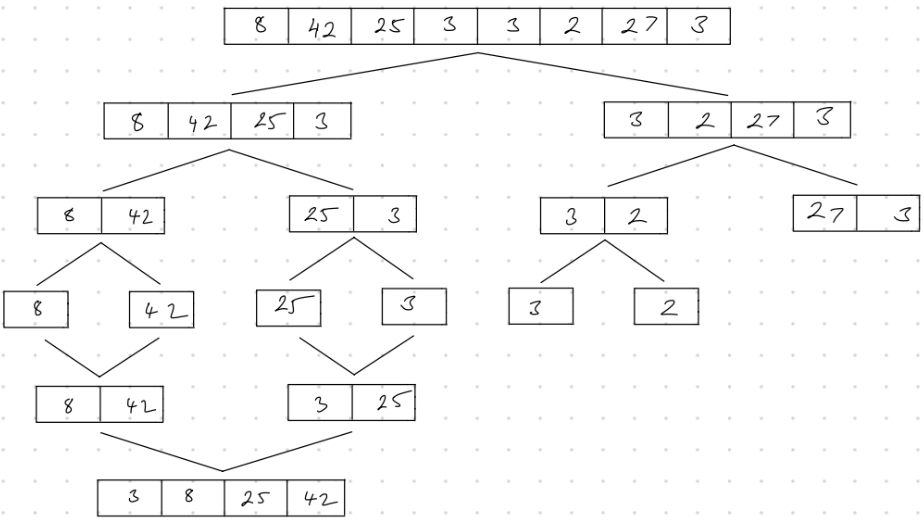
8 42 25 3 3 2 27 3
8 42 25 3
3 2 27 3
8 42
25 3
3 2
8
42
25
3
3
8 42
3 25
3 8 25 42

15



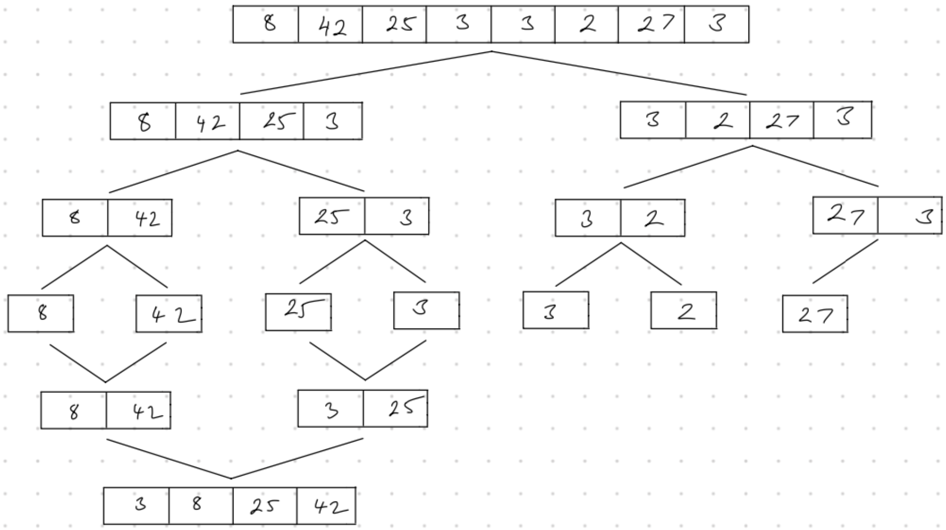
8 42 25 3 3 2 27 3
8 42 25 3
3 2 27 3
8 42
25 3
3 2
8
42
25
3
3
2
8 42
3 25
3 8 25 42

16



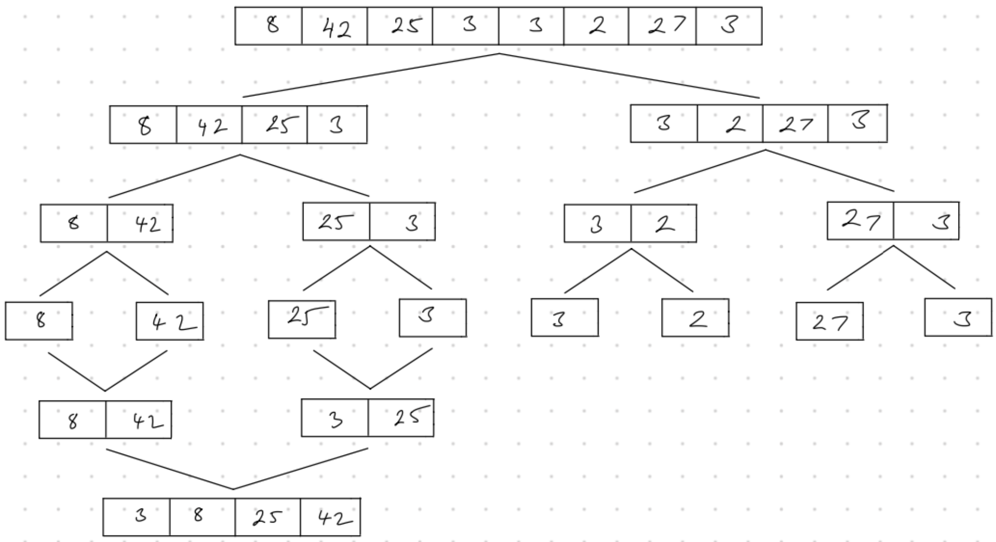
8 42 25 3 3 2 27 3
8 42 25 3
3 2 27 3
8 42
25 3
3 2
27 3
8
42
25
3
3
2
8 42
3 25
3 8 25 42

17



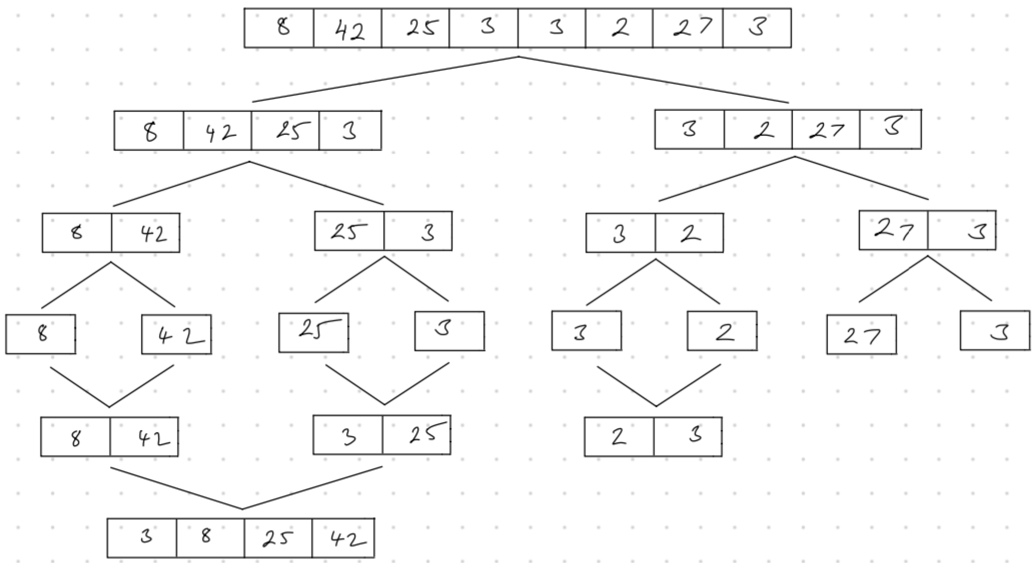
8 42 25 3 3 2 27 3
8 42 25 3
3 2 27 3
8 42
25 3
3 2
27 3
8
42
25
3
3
2
27
8 42
3 25
3 8 25 42

18



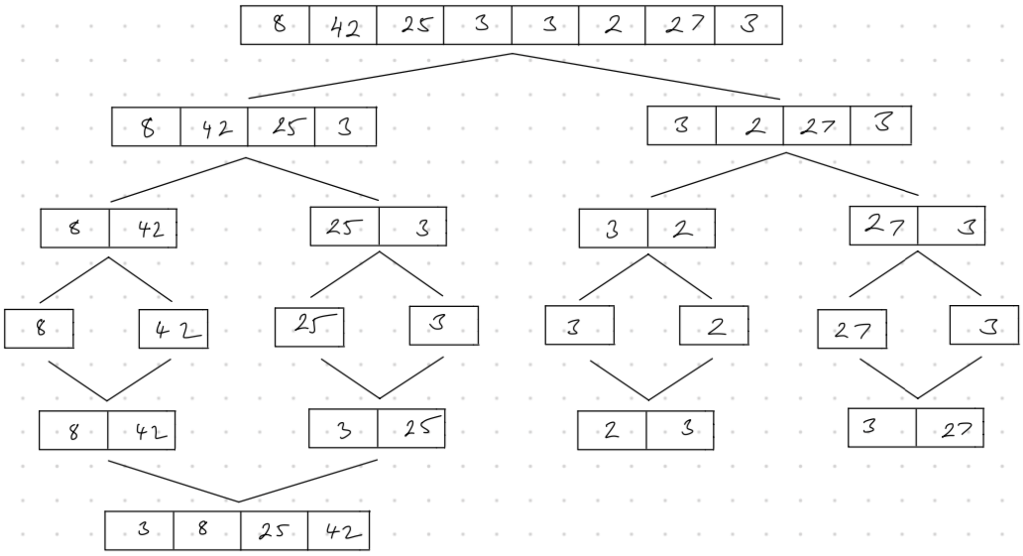
8 42 25 3 3 2 27 3
8 42 25 3
3 2 27 3
8 42
25 3
3 2
27 3
8
42
25
3
3
2
27
3
8 42
3 25
3 8 25 42

19

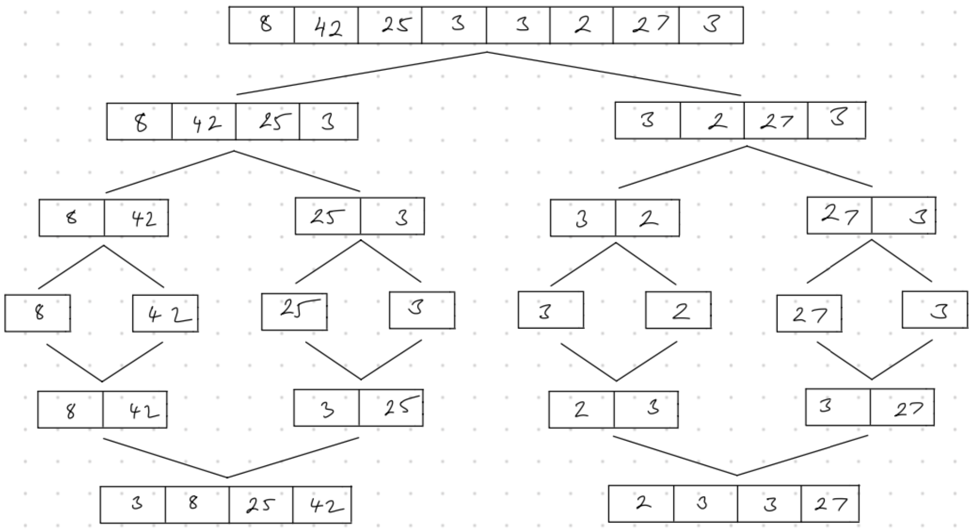


8 42 25 3 3 2 27 3
8 42 25 3
3 2 27 3
8 42
25 3
3 2
27 3
8
42
25
3
3
2
27
3
8 42
3 25
2 3
3 8 25 42

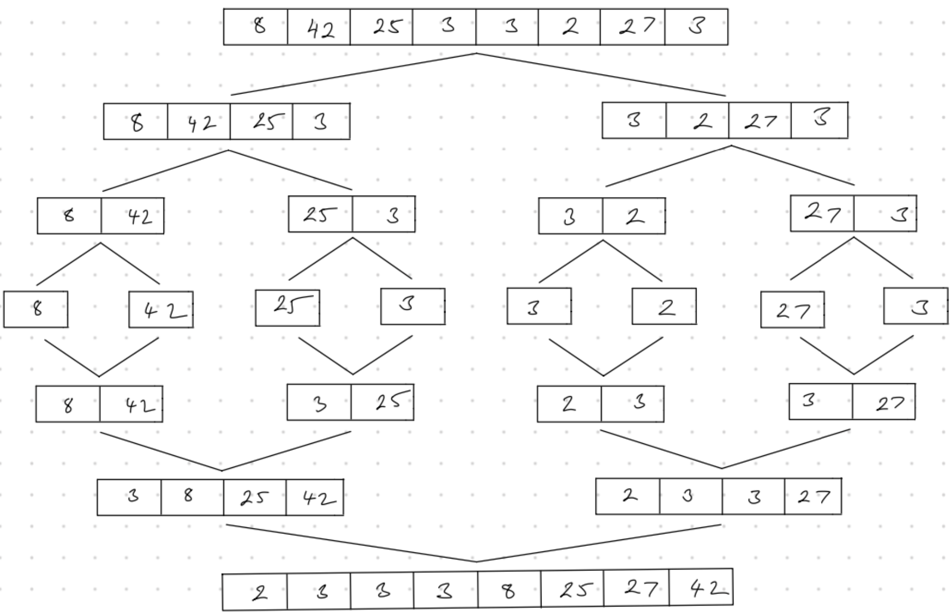
20


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

21


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

22


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

4. To sort the given vector, 22 steps are done, and this is consistent with the complexity of O(n*logn). This is because the sorting is done in two phases, the splitting phase, then the merging phase. During the splitting phase the array of 8 is recursively divided until each sub-array contains only one element. This has a complexity of O(log(n)), which in this case would be log(8) = 3. Then, during the merging phase, after all the sub-arrays have been divided, they are combined in a sorted manner. This contributes to the complexity with O(n), meaning the complexity of the algorithm is O(n*log(n)). In this case, that means that the worst-case would have 28 steps, and since the number done in this algorithm is less than or equal to that, it is consistent with the complexity.