Explanation

Task

1(a): first of all in the first line of input total element of array and we ~~[illegible]~~ set a target for the array the ~~[illegible]~~ set a flag. Then we will iterate two loops at a time, ~~[illegible]~~ that's make it time complexity $O(n^2) \Rightarrow O(n) * O(n)$. and last if/else condition for we ~~[illegible]~~ only need one output.

1(b) Count is a variable that keeps track of swapping, after each successful addition it will increment. if there is no sucessful addition the variable will remain 0 and the loops break and it will reduce time and the time complexity will be $O(n)$

2 (a) : At first we converted ~~two~~ output into two different list and then created a new list called (new_arr), Then we sorted the new list using mergesort and the time complexity of mergesort is $O(n\log n)$.

2(b) The merge function takes two list and merges them in a single sorted list by comparing them each other. The smaller element appended to the "merged" list and the corresponding list is updated by removing element. Finally the function returns the merged list with any remaining element from a and b.

Task 3: In task 3 we used merged sorting algorithm in merge sort function we are pushing ~~[illegible]~~ our "arr1" as arr and it will break the array from ~~[illegible]~~ middle as left part and right path and return in the merge function this will continue until each element separated and complete a sorted list.

Time complexity of the code is $O(n \log n)$

Task 4:

In this we have to get the ~~maxim~~ max element from the list. here, we will use a merge sort function. If the length of array $<= 1$, that's means the list have only one element and that'll be its maximum element so we will return it. two variable n1 is the left max and right one is the right max after comparing we will return the max element of the list.

Time complexity of this code is $O(n \log n)$