Q4.4) Let *A* be an array, where each of the *n* elements is a randomly chosen digit between 0 and 9. For example, if *n*=12, this array could be *A*=[3,5,1,0,5,7,9,2,2,8,8,6].
Determine whether Counting Sort or Merge Sort sorts this array faster.

Given array A = [3,5,1,0,5,7,9,2,2,8,8,6].

Algorithm and Explanation - Count Sort

Count Sort is a sorting technique based on keys between a specific range. It works by counting the number of objects having distinct key values (kind of hashing). Also, counting sort makes assumptions about the data, for instance, it assumes that values are going to be in the range of 0 to 10 or 10 – 99 etc, Some other assumptions counting sort makes are input data will be all real numbers. Like other algorithms, this sorting algorithm is not a comparison-based algorithm, it hashes the value in a temporary count array and uses them for sorting.

```
function countSort(A):
    output = [0 for i in range(len(A))]
    count = [0 for i in range(9)]
    ans = ["" for _ in A]
    for i in A:
        count[ord(i)] += 1
    for i in range(9):
        count[i] += count[i-1]
    for i in range(len(A)):
        output[count[ord(A[i])]-1] = A[i]
        count[ord(A[i])] -= 1
    for i in range(len(A)):
        ans[i] = output[i]
    return ans
```

In the given question, n array elements are randomly chosen digit between 0 and 9. This means that the array elements are arranged in any order and the range of the

elements vary from 0 to 9. In this question, analyzing the running time complexity of the countSort algorithm would give is,

- First for loop is executed for n times and hence it takes O(n) time.
- Second, for loop is executed for k times and hence it takes O(k) time.
- Third, for loop is executed for n times and hence it takes O(n) time.
- Fourth, for loop is executed for n times and hence it takes O(n) time.

Where n is the number of the elements in the array and k is the range of elements (largest-smallest), which in this question 9-0 = 9. The basic intuition behind this can be that, as counting the occurrence of each element in the input range takes k time and then finding the correct index value of each element in the sorted output array takes n time, thus the total time complexity becomes O(n+k).
Furthermore, Count Sort algorithm is a non-comparison based sorting algorithm, that is, the arrangement of elements in the array does not affect the flow of algorithm. No matter if the elements in the array are already sorted, reverse sorted or randomly sorted, the algorithm works the same for all these cases and thus the time complexity for all such cases is same which is O(n+k).

Hence, the overall time complexity of Count Sort in this question in O(n+k), that is linear time. We can substitute k as 9, in this question.

Algorithm and Explanation - Merge Sort

The Merge Sort algorithm is a divide and conquer sorting algorithm. So, in this algorithm, the array is initially divided into two equal halves and then they are combined in a sorted manner. We can think of it as a recursive algorithm that continuously splits the array in half until it cannot be further divided. This means that if the array becomes empty or has only one element left, the dividing will stop, i.e. it is the base case to stop the recursion. If the array has multiple elements, we split the array into halves and recursively invoke the merge sort on each of the halves. Finally, when both the halves are sorted, the merge operation is applied. Merge operation is the process of taking two smaller sorted arrays and combining them to eventually make a larger one.

MergeSort algorithm follows the following steps,

- Declare left variable to 0 and right variable to n-1
- Find mid by medium formula. mid = (left+right)/2

- Call merge sort on (left,mid)
- Call merge sort on (mid+1,rear)
- Continue till left is less than right
- Then call merge function to perform merge sort.

Since we repeatedly divide the (sub)arrays into two equally sized parts, if we double the number of elements n, the number of division stages is logn. The merge process does not contain any nested loops, so it is executed with linear complexity. If the array size is doubled, the merge time doubles, too. The total time is, therefore, the same at all merge levels. So we have n elements times logn division and merge stages. Therefore, the time complexity is O(nlogn). And that is regardless of whether the input elements are pre-sorted or not. Merge Sort is therefore no faster for sorted input elements than for randomly arranged ones.
Merge Sort is a recursive algorithm and time complexity can be expressed as following recurrence relation $T(n) = 2T(n/2) + \Theta(n)$.
The above recurrence can be solved either using the Recurrence Tree method or the Master method. It falls in case II of Master Method and the solution of the recurrence is $\Theta(nlogn)$. The running time complexity of Merge Sort is $\Theta(nlogn)$ in all 3 cases (worst, average and best) as merge sort always divides the array into two halves and takes linear time to merge two halves.

Hence, we come to the conclusion that the average case time complexity for MergeSort algorithm in this question will be O(nlogn).

Hence, for the given question, where A consists of *n* elements where each of the elements is a randomly chosen digit between 0 and 9, Count Sort works better and sorts this array faster in O(*n+k*) time as compared to Merge Sort which takes O(*nlogn*) time.