**CS-317 Project-1 Report**

The observations of this project are as follows:

1. Method-2 yields better(lesser) count than method-1.
2. By comparing them theoretically, firstly Quicksort has an order of growth of O(nlg(n)) for best and average cases. O(n2) for worst cases.

* For Method-1, we have converted the 2-D matrix of (n\*n) into a 1-D matrix, so we have a total of n2 elements. So, theoretically the order of growth for the given example is **50625**(n=225) for worst case, for the best case and average case the order of growth is **1758.1**
* For Method-2, we have sorted the matrix row-wise and column-wise, so we should calculate the order of growth for every row and column i.e. n=15, Best case and average case= 15lg (15) for each row. So, we have to do it for rows and columns (=30). So, the overall best and average case for Method-2 is **1758.1**, which is same as Method-1’s Best and average case. Now, the worst case for method-2 is **6750**

1. By comparing them practically we can say that method-2 has better count than method1. Because in method-1 we are converting the entire matrix into a single array we have more number of elements and thus we are forced to use the black box more number of times than method-2 where we just divide the matrix into rows and columns and Quicksort is applied for a row or a column at a particular call. Thus, we use the black box comparatively less number of times.
2. For example, we have taken a 15\*15 matrix (as shown in fig 1) in this project and after sorting it using method-1 and method-2, we got a count of **5806** in method-1 and **1993** in method-2. According to the requirements the matrix should be sorted in a non-decreasing order for both row and column. So, method-2 satisfies the requirement and that too with better count.

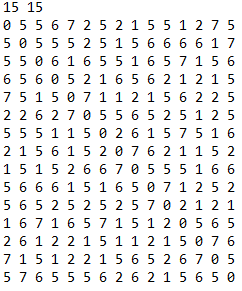
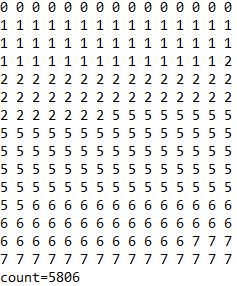
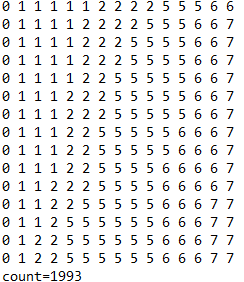
  

Fig:1 Fig:2 Fig:3