# CS 590 - Assignment 2

**Submitted By: Shweta Madhale**

**CWID: 20015921**

**Abstract**:

A class random generator is used to generate random strings up to a given length between characters ‘a’ and ‘z’. Implementation of insertion sort and counting sort to sort given string according to character at position ‘d’ was done. For the given problem statement, implementation of the radix sorting algorithm was done using the insertion sort and counting sort algorithms. Also, the runtime for each of the algorithm for different input sizes was observed.

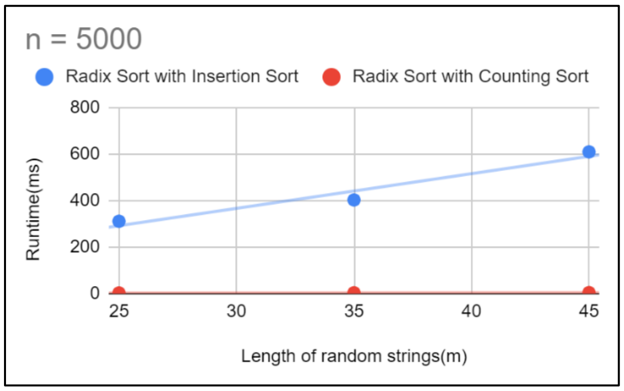
Both algorithms were tested for input size n = 5000, 6000, 7000, 8000, and length of random strings m = 25, 35, 45. The runtime for these were noted and plotted in a scatterplot.

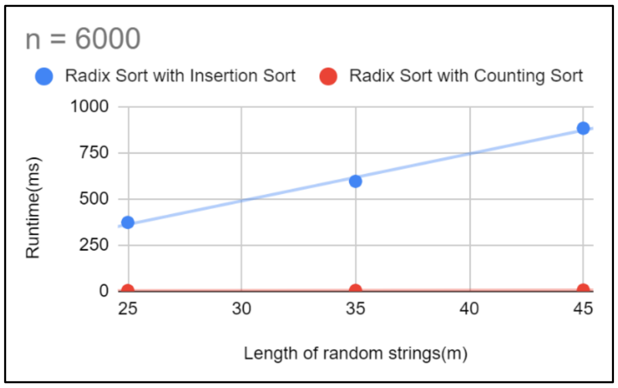
**Observations:**

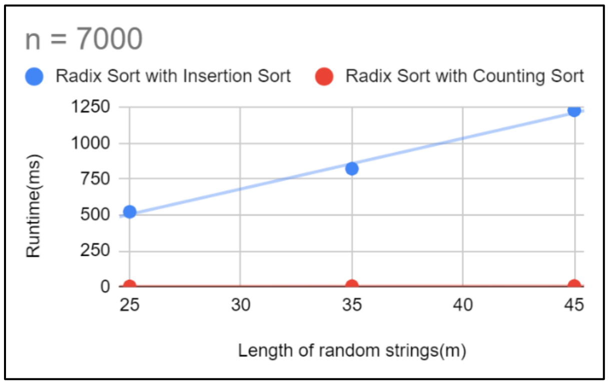
Average runtime for each input parameter corresponding to the algorithm used.

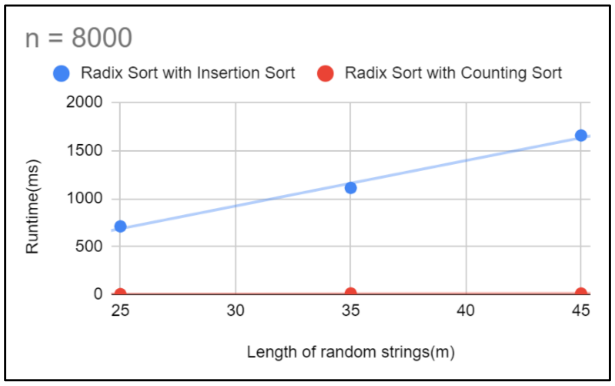
| | n = 5000 | | n = 6000 | | n = 7000 | | n = 8000 | |
|---|---|---|---|---|---|---|---|---|
| m | Radix Sort with Insertion Sort | Radix Sort with Counting Sort | Radix Sort with Insertion Sort | Radix Sort with Counting Sort | Radix Sort with Insertion Sort | Radix Sort with Counting Sort | Radix Sort with Insertion Sort | Radix Sort with Counting Sort |
| 25 | 311.2 | 2.9 | 373.2 | 3.9 | 522.6 | 4 | 709.8 | 4.3 |
| 35 | 402.8 | 3.7 | 596 | 4.6 | 822.2 | 5.8 | 1110.4 | 11.5 |
| 45 | 609.6 | 4.4 | 883.6 | 6.7 | 1228.2 | 6.6 | 1656.8 | 10.6 |

**Discussion:**









The plots of the runtime vs the input size parameters are shown above. From the plots above, the radix sort implemented using insertion sort has a very high runtime compared to the one with counting sort for similar constraints. Insertion sort is an algorithm which performs well for smaller input constraints and worsens for higher input constraints, but when compared with counting sort it does not fair well on smaller constraints. When the input possible values are not high, counting sort is an ideal stable sort to be utilized in the radix sort routine. Radix sort using counting sort runs is linear sorting. But this version does not sort in place, which can affect the storage constraints.

**Analysis:**

i. Radix sort implemented using insertion sort has much higher runtime than the one with counting sort.
ii. Since the possible values are less, radix sort with counting sort intermediate performs exceptionally well.
iii. With increase in the input constraints, the radix sort with insertion sort shows drastic increase in the runtime, almost doubling.
iv. With increase in the input constraints, the radix sort with counting sort does not change drastically and grows gradually even with sudden increases in input constraints.
v. The radix sort with insertion sort performs an in-place sorting.
vi. The radix sort with counting sort does not perform in place sorting.
vii. The radix sort with counting sort intermediate sorts linearly.

## Conclusion:

i. After implementing radix sort with two stable intermediates viz. insertion sort, and counting sort, the performance of each of these was observed.
ii. The radix sort with insertion sort has a much higher runtime for even smaller input constraints.
iii. Also, radix sort with insertion sort sees a drastic increase in the runtime with increase in input values.
iv. In contrast, radix sort with counting sort is extremely efficient as the possible values are less.
v. The radix sort with counting sort has just gradual increase in runtime with increase in the input constraints.

vi. For the problems like that of problem statement, it is ideal to use the radix sort with counting sort as opposed to the one with insertion sort.

vii. Also, the radix sort with counting sort behaves like linearly. But it does not perform in-place sorting. This can cause problems when there are memory constraints.