CS -590: Algorithms

Homework 2-2: Linear Sorting

Poorvi Raut: 20009560

I. Radix Sort using Insertion sort

Coefficients taken are:

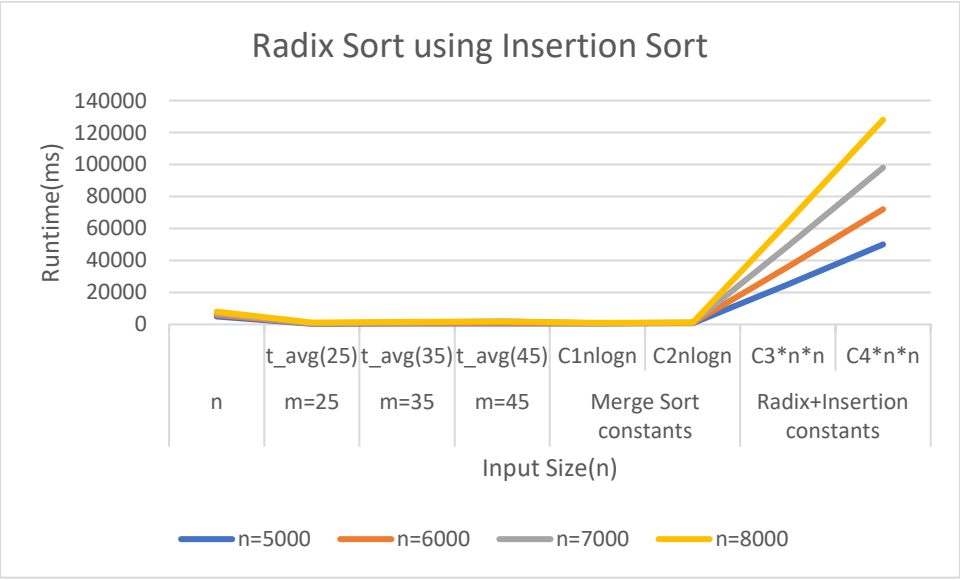
C1=0.03

C2=0.04

C3=0.001

C4=0.002

n	m=25	m=35	m=45	Merge Sort constants		Radix+Insertion constants	
	t_avg(25)	t_avg(35)	t_avg(45)	C1nlogn	C2nlogn	C3*n*n	C4*n*n
5000	383.3	534.2	649.2	554.84	739.79	25000	50000
6000	556.8	750.4	929.7	680.06	906.75	36000	72000
7000	754.7	1085.8	1861.8	807.47	1076.62	49000	98000
8000	1084.5	1518.8	1849.4	936.74	1248.98	64000	128000



II. Radix Sort using Counting Sort

Coefficients taken are:

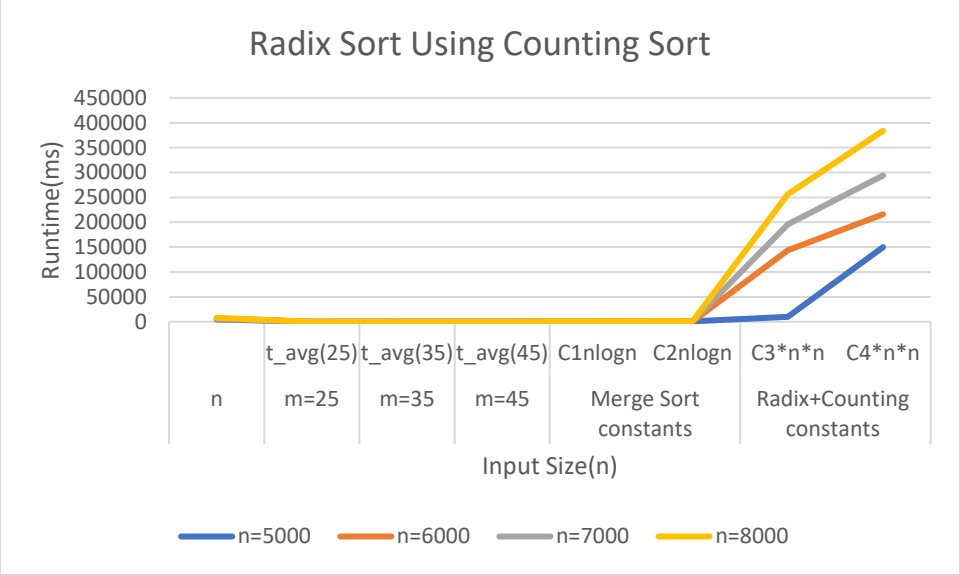
C1=0.02

C2=0.05

C3=0.004

C4=0.006

n	m=25	m=35	m=45	Merge Sort constants		Radix+Counting constants	
	t_avg(25)	t_avg(35)	t_avg(45)	C1nlogn	C2nlogn	C3*n*n	C4*n*n
5000	9.2	12.6	16.3	369.89	924.74	10000	150000
6000	10	15.3	19.7	453.37	1133.44	144000	216000
7000	13.7	18.1	25.3	538.31	1345.78	196000	294000
8000	14.7	21.3	26.2	624.49	1561.23	256000	384000



OBSERVATIONS:

Comparing Radix Sort using Insertion Sort and Counting Sort

1. Radix Sort using Insertion Sort:

Insertion Sort runtime complexity is O(n^2)

Implementing Radix sort using insertion sort runtime is slower since- Insertion sort has a runtime complexity of O(n^2), due to nested loops i.e. while loop nested in for loop).

2. Radix Sort using Counting Sort:

- Counting Sort has a runtime complexity as O(n+k) i.e., O(n) if and only if k=O(n)
- Implementing Radix sort using Counting Sort is much faster than using Insertion Sort.
- Counting Sort works well with smaller values but depends on the number of values.
- There is memory storage issue in counting sort since an additional storage is required in algorithm as we saw in the assignment must create another array.
- The runtime complexity of Radix Sort is O(d*(n+k)) where "d" is number of passes in for loop

  Radix sort with Counting sort gives much better performance than Radix sort with Insertion sort.

  For example, for m = 45, n = 8000:

  Radix sort with counting sort is 70 times (=1849.4/26.2) faster than radix with insertion sort.

CONCLUSION:

- For larger values of "n" Radix Sort becomes slower hence we need to use it for smaller values of d. It is observed that in the graph for larger values of n i.e. input size the time complexity of radix sort algorithm increases.
- When "d" gets large the runtime complexity of radix sort is worse than merge sort i.e., nlog(n)
- Time complexity comparison using both the algorithms.
  Radix +Insertion Sort: O(n^2)
  Radix + Counting Sort: O(n)

So, it is better to using Radix with Counting Sort for smaller values of n for better performance.

- Space Complexity Comparison
  Radix +Insertion Sort: For Insertion sort, space complexity is O(1) since it sorts an entire array just by using one extra variable.

Radix + Counting Sort: Counting sort has space complexity O(n + k), as two arrays needs to be created to copy the array contents while sorting.

Therefore large value of “n” has a memory issue for Counting Sort algorithm.