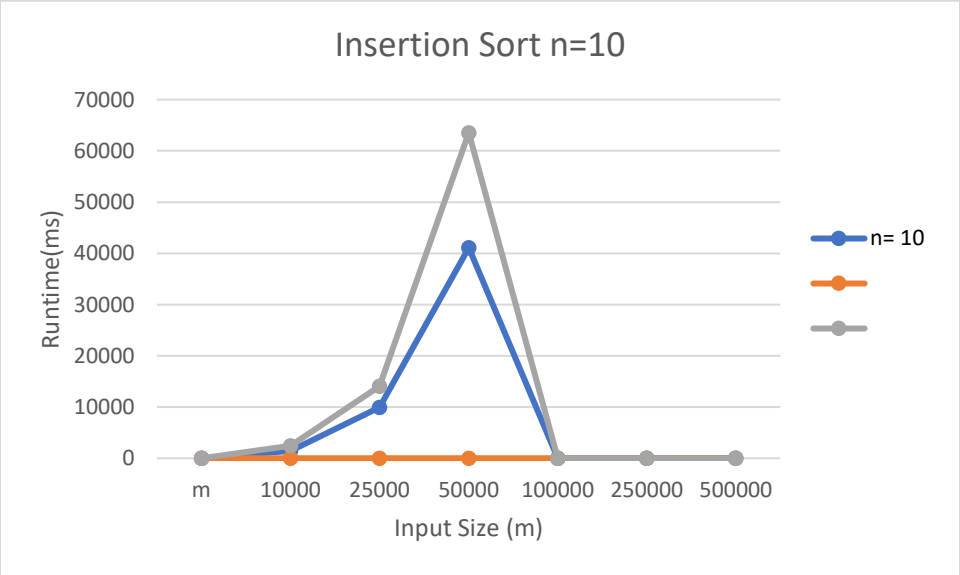
CS -590: Algorithms

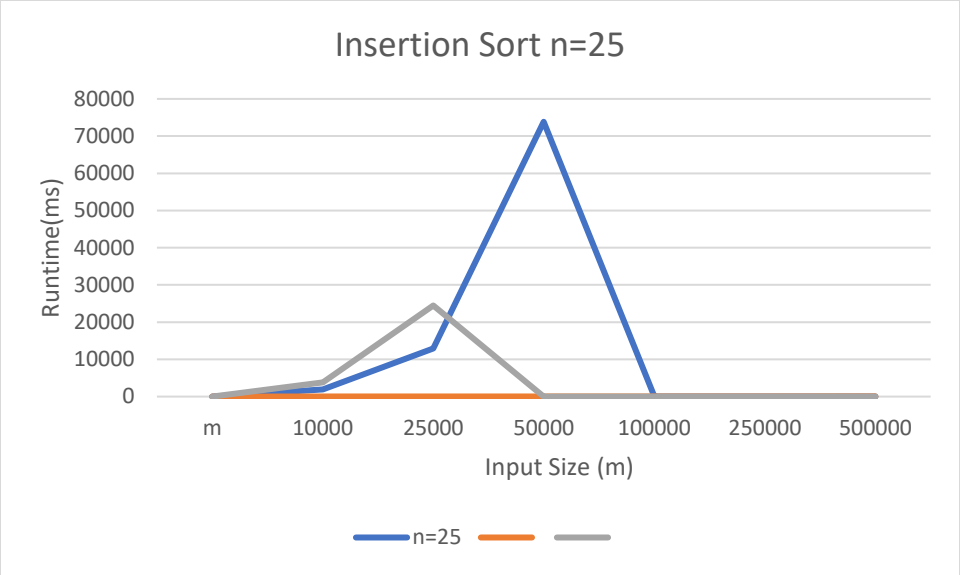
Homework 1: C++ & running times.

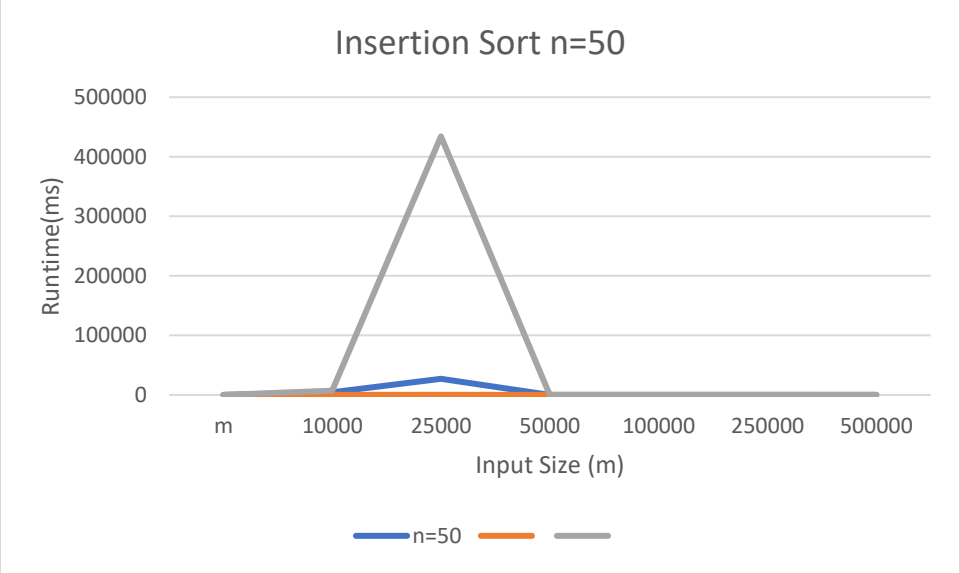
Poorvi Raut: 20009560

INSERTION SORT:

m	n= 10			n=25			n=50		
	Random	Sorted	Reverse Sorted	Random	Sorted	Reverse Sorted	Random	Sorted	Reverse Sorted
10000	1471.1	0.7	2461.4	1941	0.4	3846	4339.1	1.5	6884.7
25000	9950.5	0.16	14071.5	12904.7	1.9	24480.4	27047.5	3.6	434202
50000	41093.4	0.28	63546.3	73851.7	5.4	-	-	6.6	-
100000	-	0.2	-	-	10.9	-	-	13.3	-
250000	-	10.9	-	-	26.6	-	-	32.7	-
500000	-	22	-	-	53.0	-	-	64.9	-





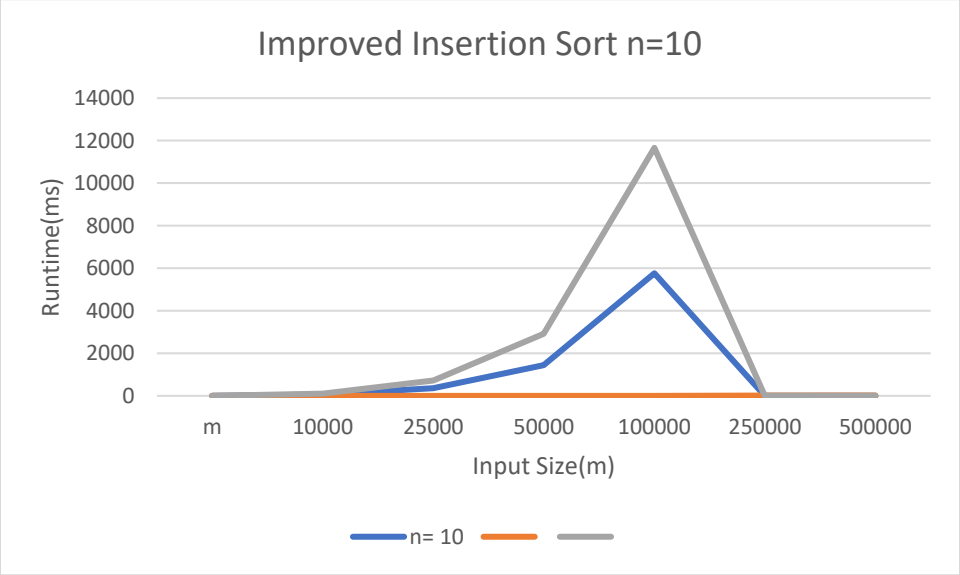


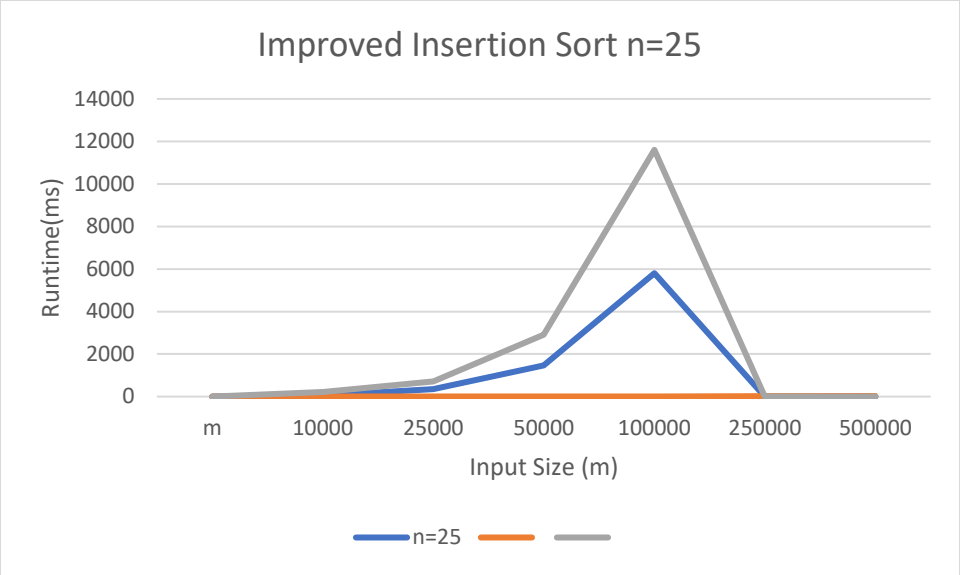
Result of computation:

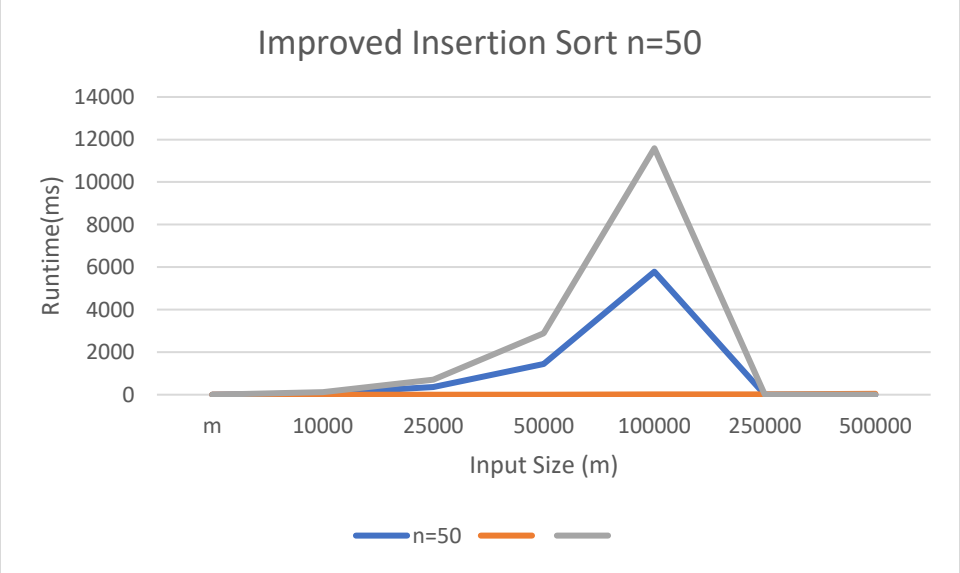
- For naïve insertion sort algorithm, in order to compute vector length every time we must compare the value of runtime vs input size (m) which takes longer time as the input size varies.
- Also, as we are computing the vector length multiple times for the same vector value, so additional time is wasted in doing so.

IMPROVED INSERTION SORT:

m	n= 10			n=25			n=50		
	Random	Sorted	Reverse Sorted	Random	Sorted	Reverse Sorted	Random	Sorted	Reverse Sorted
10000	56.7	0.3	103.1	57.4	0.6	216	56.9	0.8	113.4
25000	358.7	0.5	720.1	358.4	1.2	709.7	355.8	2.0	705.8
50000	1435.8	1.1	2915.4	1459.8	2.7	2904.3	1443.8	4.8	2890.7
100000	5761.6	2.5	11662	5802.1	5.5	11605.2	5784.9	8.2	11589
250000	-	5.9	-	-	13.9	-	-	20.3	-
500000	-	12.6	-	-	26.7	-	-	40.7	-







Result of computation:

- For improved insertion Sort algorithm, we are improving the naïve insertion sort algorithm by per computing the vector lengths and comparing the value of runtime vs input size (m) and reducing the running time.
- The algorithm runs in logarithmic running time but its asymptotic running time does not change and had time complexity as O(n^2).

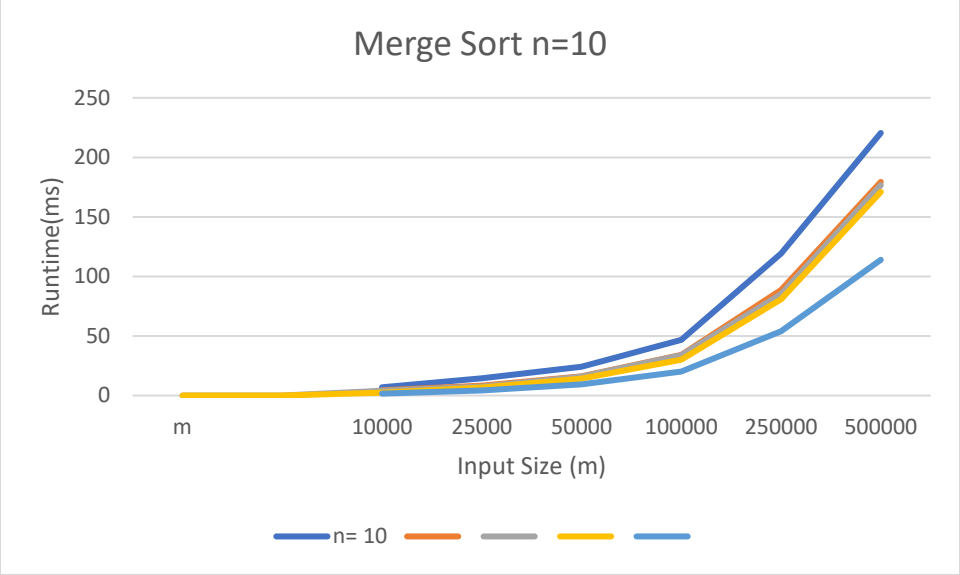
MERGE SORT:

Co-efficients taken are:

C1: 0.00006

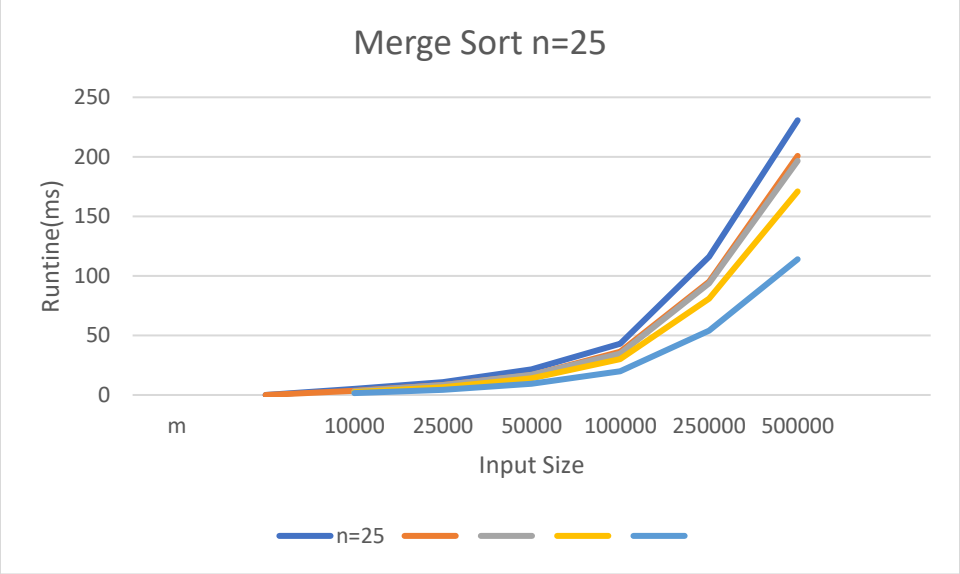
C2: 0.00004

m	n= 10				
	Random	Sorted	Reverse Sorted	C1 *m*Log(m)	C2 *m*log(m)
10000	6.9	4.4	3.9	2.4	1.6
25000	14.5	8.2	7.5	6.59	4.39
50000	24.1	16	15.8	14.09	9.39
100000	46.7	34	33.7	30	20
250000	119.2	88.7	85.2	80.96	53.97
500000	220.5	179.4	176.7	170.96	113.97



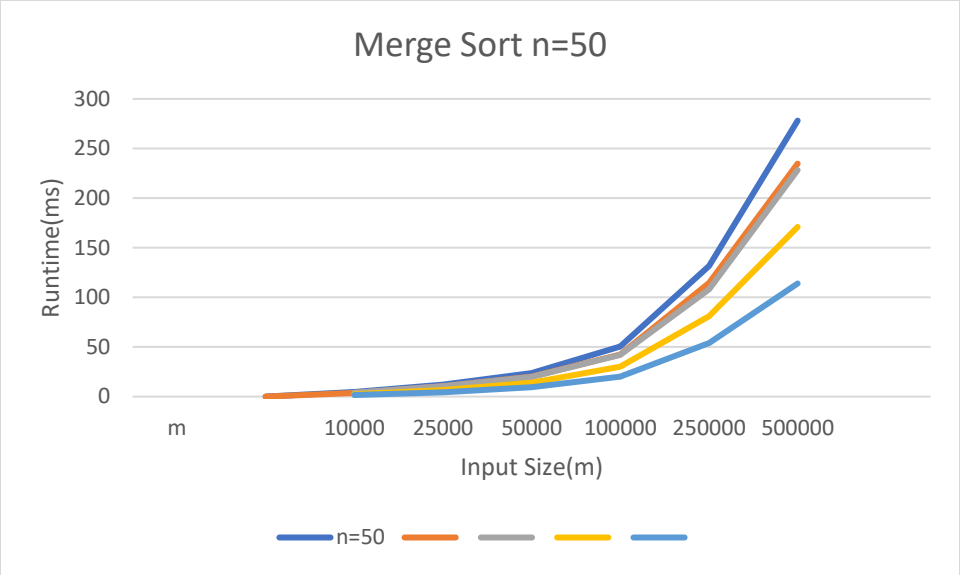
Merge Sort n=10 (Linear y axis)

m	n=25				
	Random	Sorted	Reverse Sorted	C1 = 0.00006	C2 = 0.00004
10000	5	3.6	3.2	2.4	1.6
25000	10.7	8.3	8.3	6.59	4.39
50000	21.5	16.8	17	14.09	9.39
100000	42.9	36.3	34.8	30	20
250000	116.1	95.1	93.7	80.96	53.97
500000	230.7	200.8	196.6	170.96	113.97



Merge Sort n=25 (Linear y axis)

m	n=50				
	Random	Sorted	Reverse Sorted	C1 = 0.0000	C2 = 0.00004
10000	4.6	3.8	3.9	2.4	1.6
25000	12	10.3	10.3	6.59	4.39
50000	23.8	20.1	19.7	14.09	9.39
100000	50.4	42.3	41.9	30	20
250000	131.6	114.7	108.1	80.96	53.97
500000	278.1	234.8	228.3	170.96	113.97



Merge Sort n=5 (Linear y axis)

OBSERVATIONS:

Discussing and comparing the worst case /Best case/ Average Case of Naïve Insertion sort, Improved Insertion sort and Merge sort:

1. **Randomly sorted array – Average case**
   Time complexity of Naïve and Improved insertion sort is O(N^2) and is similar to worst case performance.

2. **Sorted array – Best case:**

- In Insertion sort and Improved insertion sort – since the array is already sorted, the inner while loop will never execute as the while loop condition will never be satisfied. Hence, time complexity stays at O(n). (Only the outer for loop executes). The time taken to sort an already sorted array is very low, for lower input size m and n.
- Merge sort best case time complexity is O(n*log(n)). It is same as worst case and average case, since for all cases, the array must be divided, conquered and merged recursively.

3. **Inverse sorted array – Worst case:**

For an inverse/reverse sorted array, the fastest algorithm is Merge sort. - Since Insertion sort algorithm has 2 loops: 1. for loop, and a while loop nested within the for loop, it has a time complexity of O(n^2) in worst case.

- Merge sort does not have nested for loops, and works with Divide-and-Conquer-algorithm, and hence divides the input into 2 subproblems and conquers each subproblem and merges recursively. Hence, the time complexity is O(N*log(n)), even for worst case.