CS 590 – Algorithms – Assignment 1

Insertion Sort :

Time Complexity Analysis for insertion sort algorithm is O(N) in the best case that is when the input vector is passed in the sorted fashion. On the other hand, the time complexity of the insertion sort is O(N^2) in the worst case scenario that will be when the vector is passed into the algorithm in reverse sorted order which can be spelled as Inversely sorted Vector.

Lets run some cases where we have random integer vectors ranging from 10,000 to 1,00,000 with the dimensions between the factors of 10, 25 and 50 to gain deeper perspective in the theoritical analysis of time complexity for naïve insertion sorting algorithm.

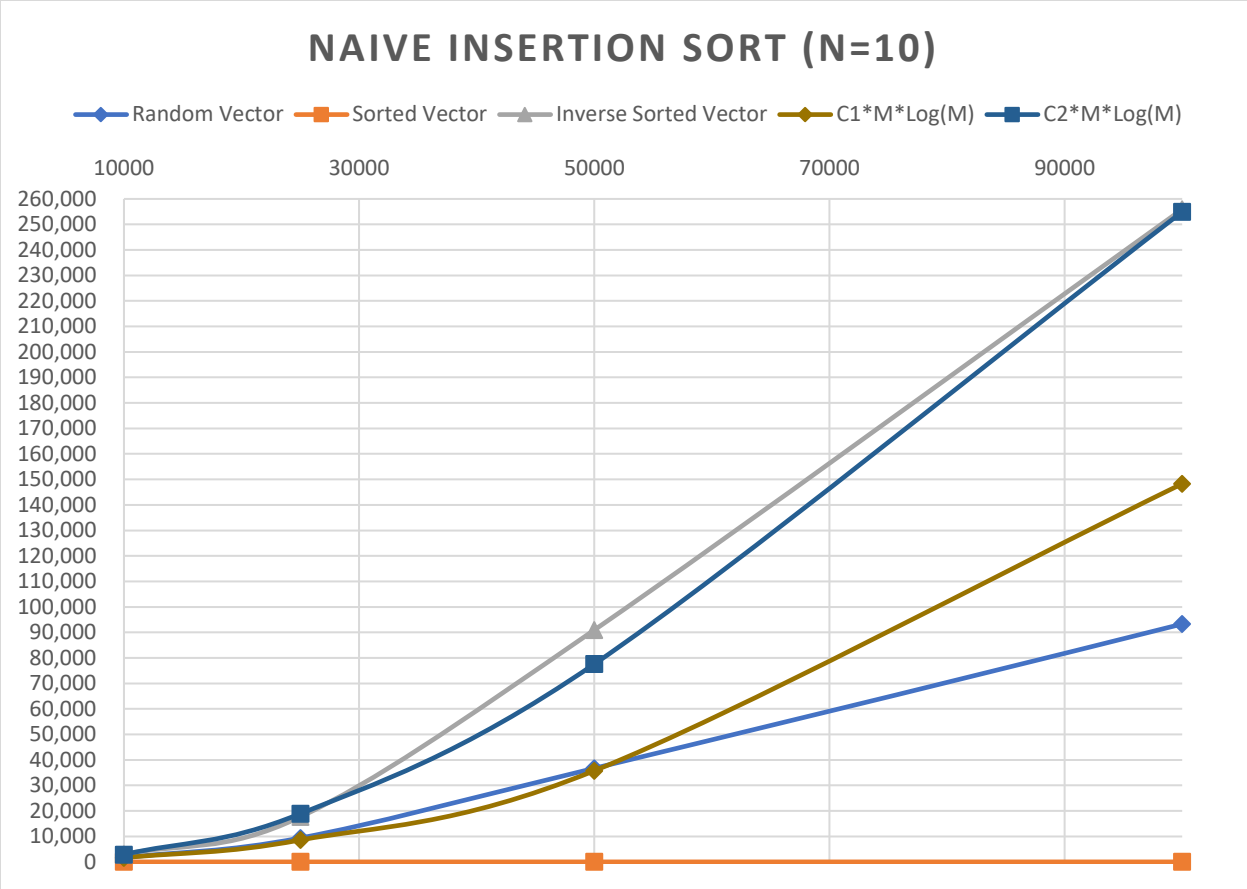
To gain more definitive knowledge about the complexity of the naïve insertion sort we will be running the sort in 3 Different category such as Random Vector, Sorted Vector And Inverse Sorted Vertor. This will help us gain graphical representation of the runtime verse dimensions for each category and eventually provide Best , Worst and probably average case of time analysis.

For the first case of naïve insertion sort:

- Dimensions : 10
- Random Vectors range : 10,000 to 1,00,000

N=10

M	Random Vector	Sorted Vector	Inverse Sorted Vector
10000	1665.28	1	2896.27
25000	9358.1	1.25	17657.45
50000	36698.36	2	90928.37
100000	93282	4.6	255935

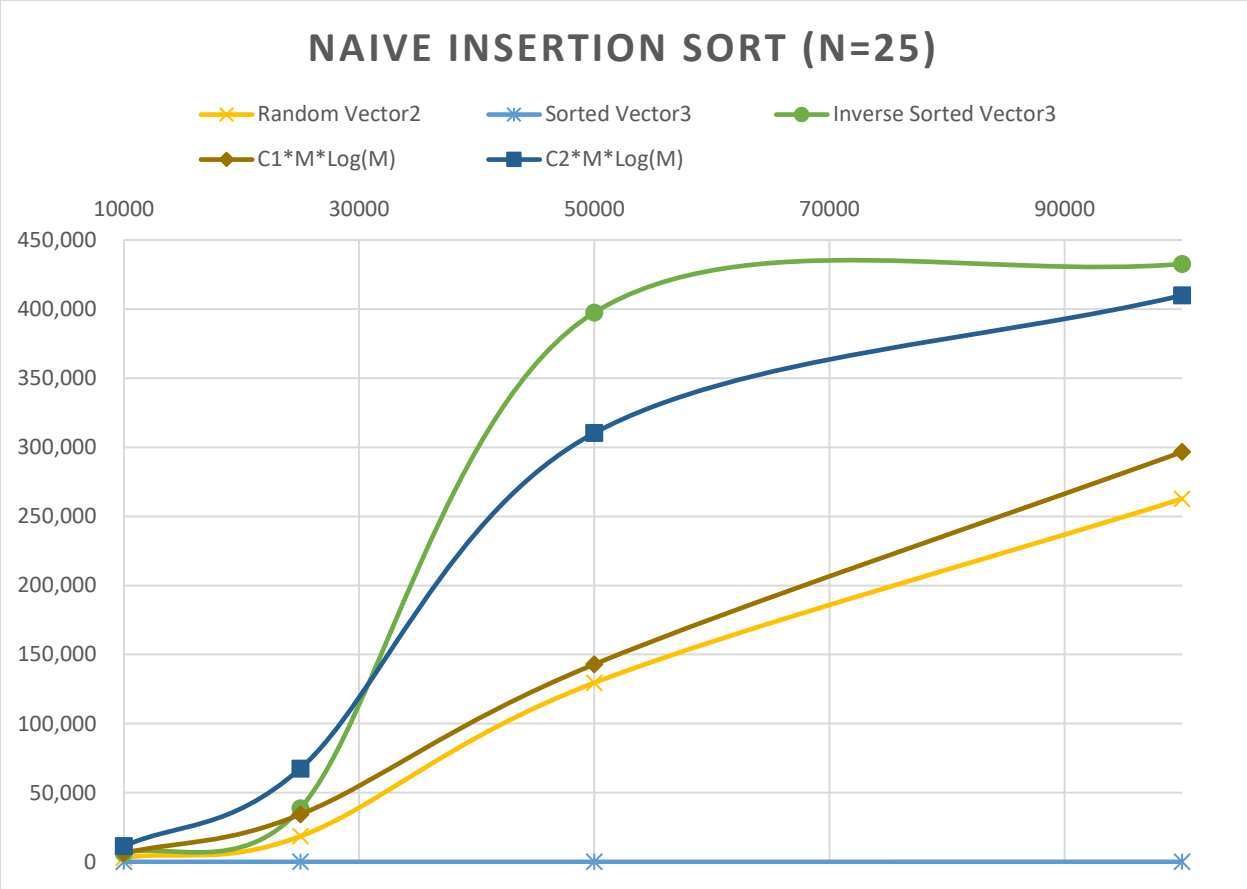


For the next iteration:

- Dimensions : 25
- Random Vectors range : 10,000 to 1,00,000

N=25

M	Random Vector	Sorted Vector	Inverse Sorted Vector
10000	3128.45	1.27	6364.55
25000	18492.36	4.28	38672.91
50000	129557.82	6.91	397465.73
100000	262735.4	13.3	432640.4

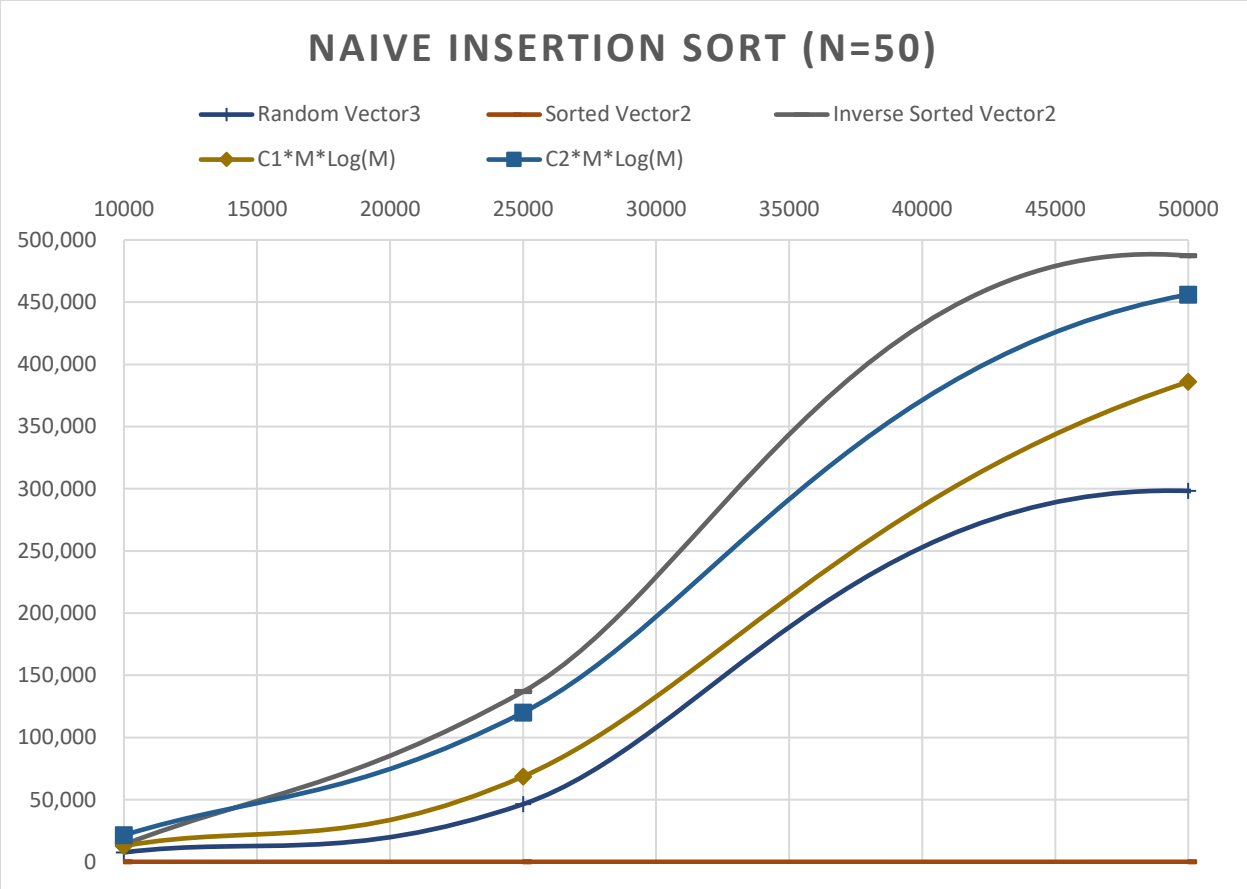


For the last case of the dimensions:

- Dimensions : 25
- Random Vectors range : 10,000 to 1,00,000

N= 50

M	Random Vector	Sorted Vector	Inverse Sorted Vector
10000	7606.36	2.64	13922
25000	46387.91	7.36	136843
50000	298207.21	18.54	487259.37
100000	-	31.36	-



Conclusion:

This naïve insertion sort follows the analysis of the theoretical time complexity. Secondly it ran out of dimensionality above the space of 1,00,000 random integer vectors.

To improve the naïve insertion sort we can make some changes in the algorithm that can scale the time complexity of the algorithm by presuming the values in the array and then redundantly sorting the array. This process will scale down our time complexity but might not gain much distinguished outcome in the asymptotic time complexity analysis of the method.

Improvised Insertion Sort:

We must run the proposed algorithm to gain the insights on the time complexity factor which could be visualized by the graphs on the similar dimensions and random vector length.

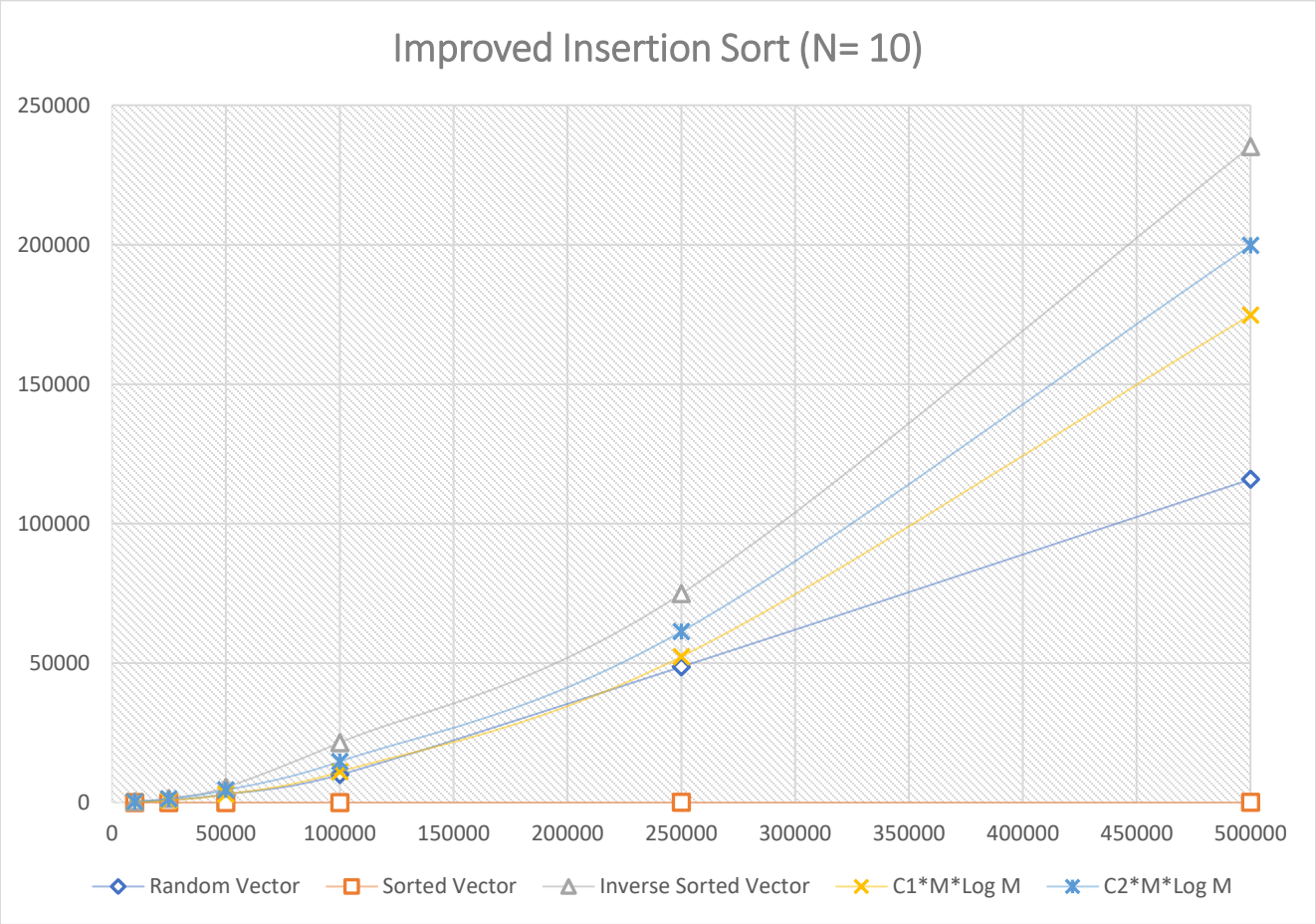
The improvised insertion sort algorithm will now be tested against the three iterations of the dimensions :

N=10,25,& 50. In the randomized vectors range of 10,000 till 5,00,000

Case 1 : N=10

Since We tested more than 1,00,000 random vectors we can claim that the improvised algorithm works much better then the naïve insertion sort that ran out of time complexity analysis.

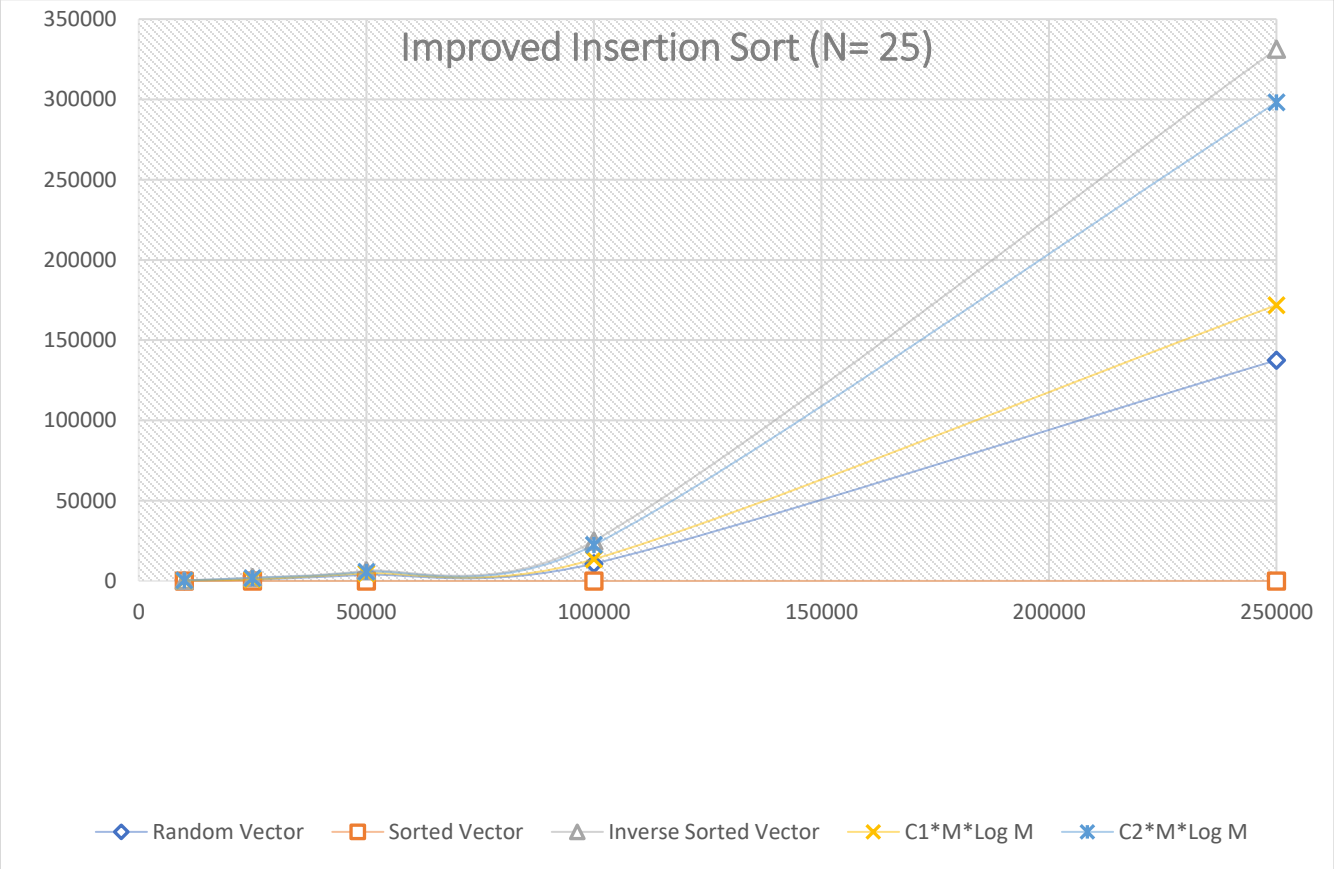
M	Random Vector	Sorted Vector	Inverse Sorted Vector
10000	152	1	327
25000	836.6	1.3	1447.6
50000	2968.8	2	5455
100000	9950.67	3.8	21467.7
250000	48635.3	7.8	59856.25
500000	115870.6	11.63	235159.67



For the next iteration: N=25

We can view that in this iteration the N is 25 for that the inverse sorted vector ran out of the time complexity analysis.

M	Random Vector	Sorted Vector	Inverse Sorted Vector
10000	184.83	1.17	402.4
25000	817.2	2.16	2069.45
50000	3828.8	6.21	6361.34
100000	10830.72	11.75	24902.25
250000	137492.76	21	331297.46
500000	476078.8	32	-

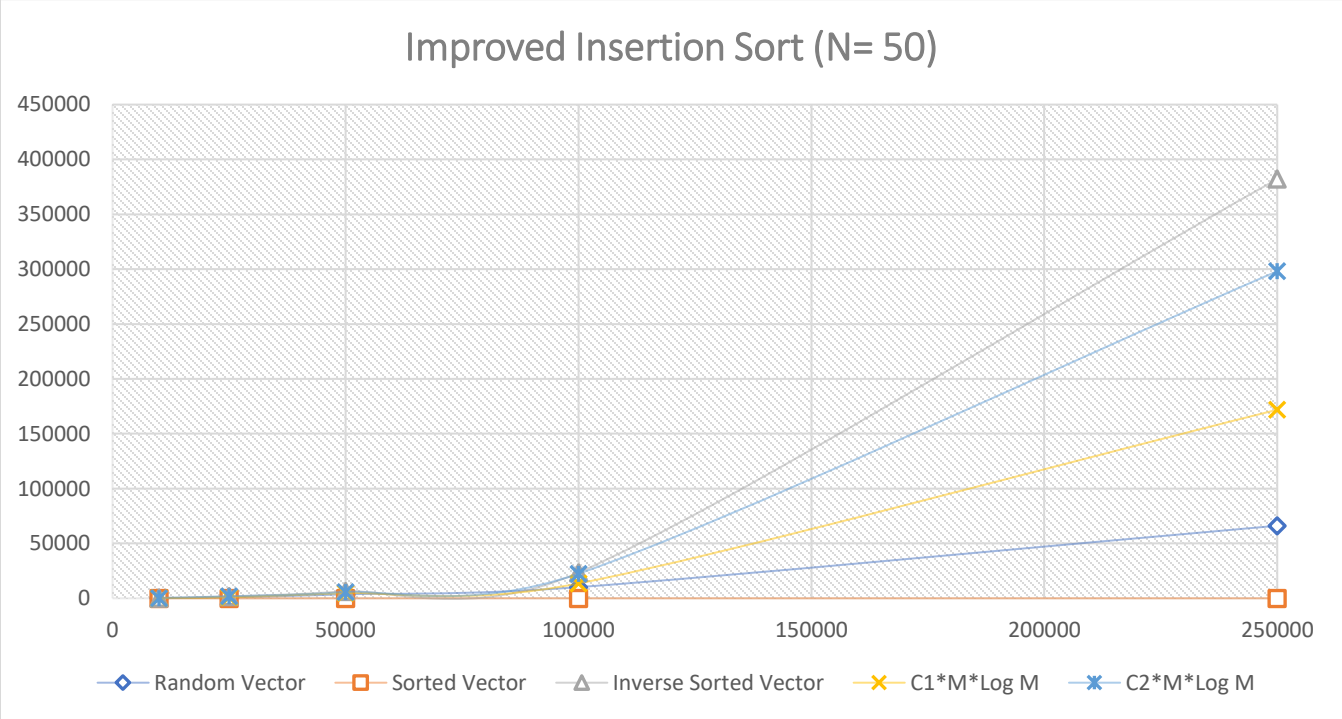


For the next iteration: N=50

We can view that in this iteration the N is 50 for that the inverse sorted vector ran out of the time complexity analysis.

N=50

M	Random Vector	Sorted Vector	Inverse Sorted Vector
10000	205	1.4	331.2
25000	929.83	5.17	1709.2
50000	3656	8.75	6426.25
100000	10188.15	16.75	23374
250000	66157.67	27.67	382160.4
500000	118367.33	44.2	-



Conclusion:

This is scaling down the naïve insertion sort algorithm and tending to lean the time complexity towards N*Log N but due to asymptotic time analysis the algorithm is following absurd time complexity.

Merge Sort Time Complexity Analysis:

It is one of the famous recursive algorithm and this algorithm is usually expressed by the time complexity of T(n) = 2*T(n/2) + O(n). This recursive relation can be further simplify to gain the solution of Big-O (n*log n). The most unique part about the technique is that it generally follows the similar time complexity in the case of Best, Average or Worst utilizing divide and conquer techniques.

Lets run some cases where we have random integer vectors ranging from 10,000 to 10,00,000 with the dimensions between the factors of 10, 25 and 50 to gain deeper perspective in the theoritical analysis of time complexity for merge sorting algorithm.

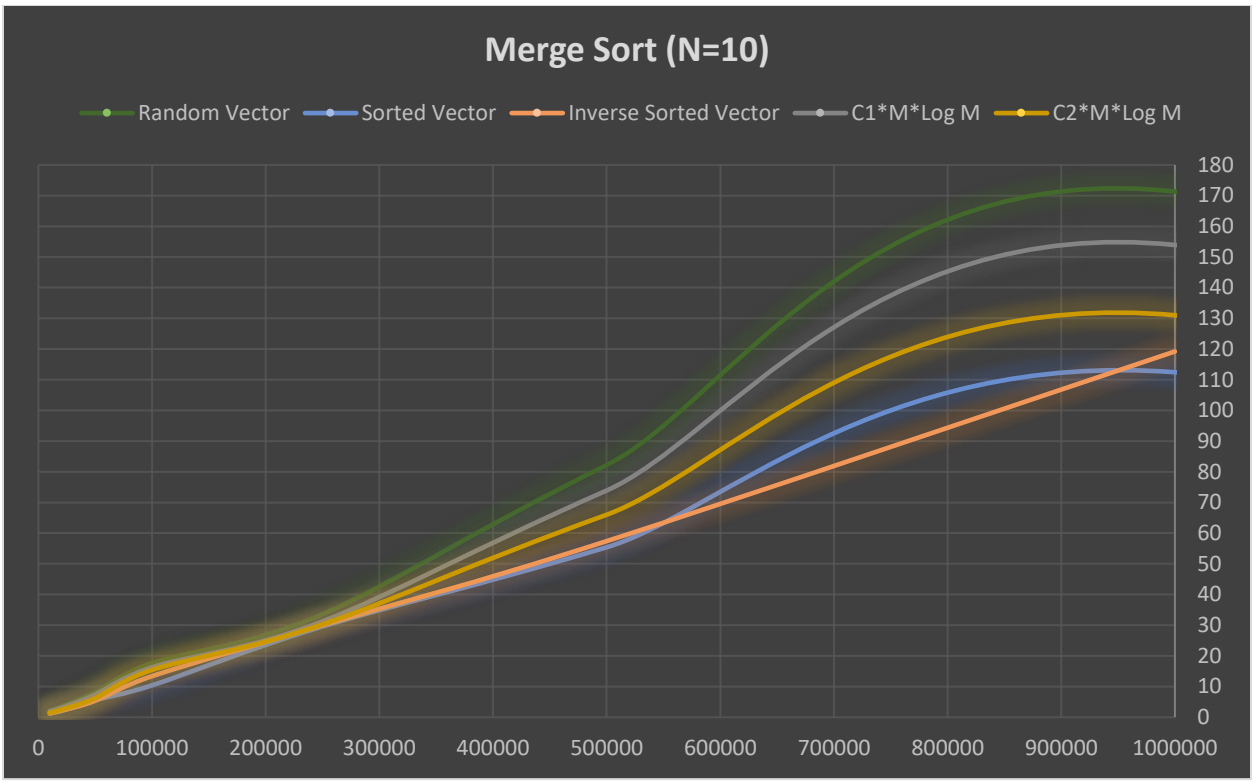
To gain more definitive knowledge about the complexity of the naïve insertion sort we will be running the sort in 3 Different category such as Random Vector, Sorted Vector And Inverse Sorted Vertor. This will help us gain graphical representation of the runtime verse dimensions for each category and eventually provide Best , Worst and probably average case of time analysis.

For the first case of Merge sort:

- Dimensions : 10
- Random Vectors range : 10,000 to 10,00,000

N=10

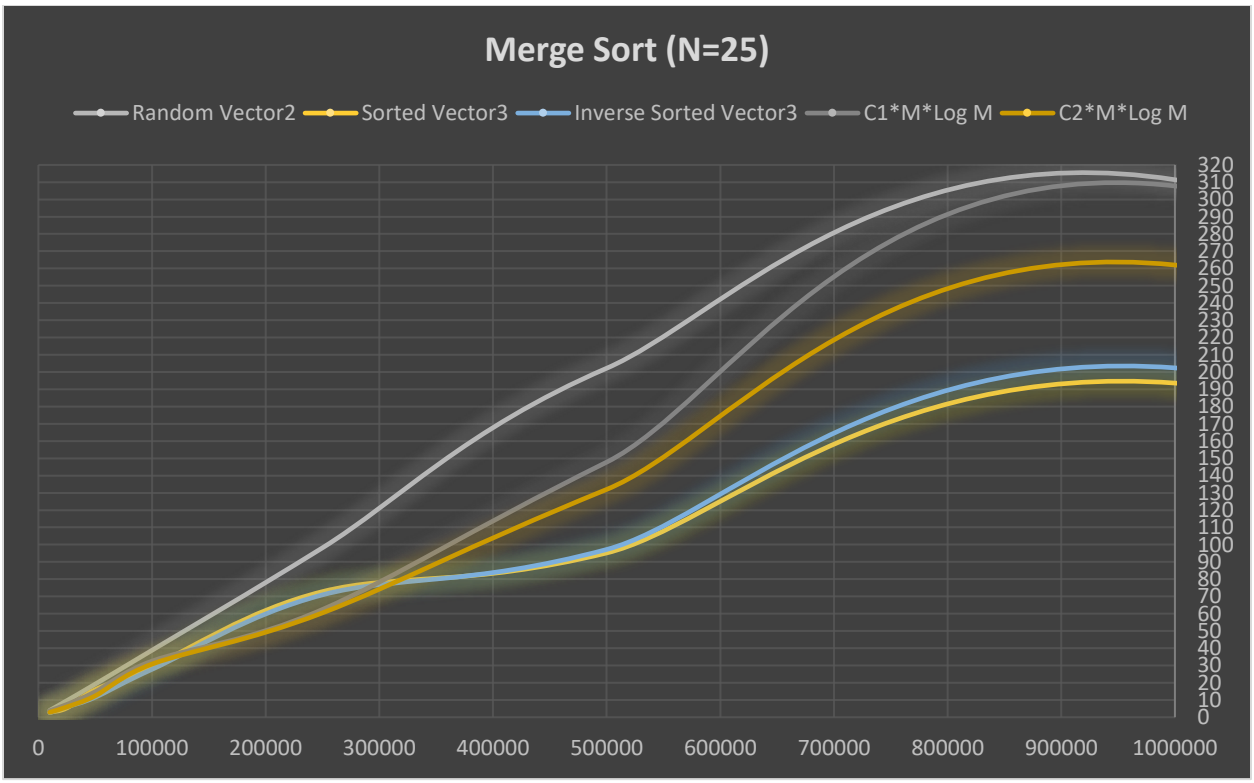
M	Random Vector	Sorted Vector	Inverse Sorted Vector
10000	2.13	1.25	1.25
25000	4.13	2.6	2.6
50000	8.2	5.6	5.4
100000	17.67	10.4	13.4
250000	33.6	29.6	30
500000	82.2	55.4	57.4
1000000	171.3	112.43	119.2
2500000	-	-	-



Lets now check for the next iterative of the merge sort using the new dimensions:

N=25

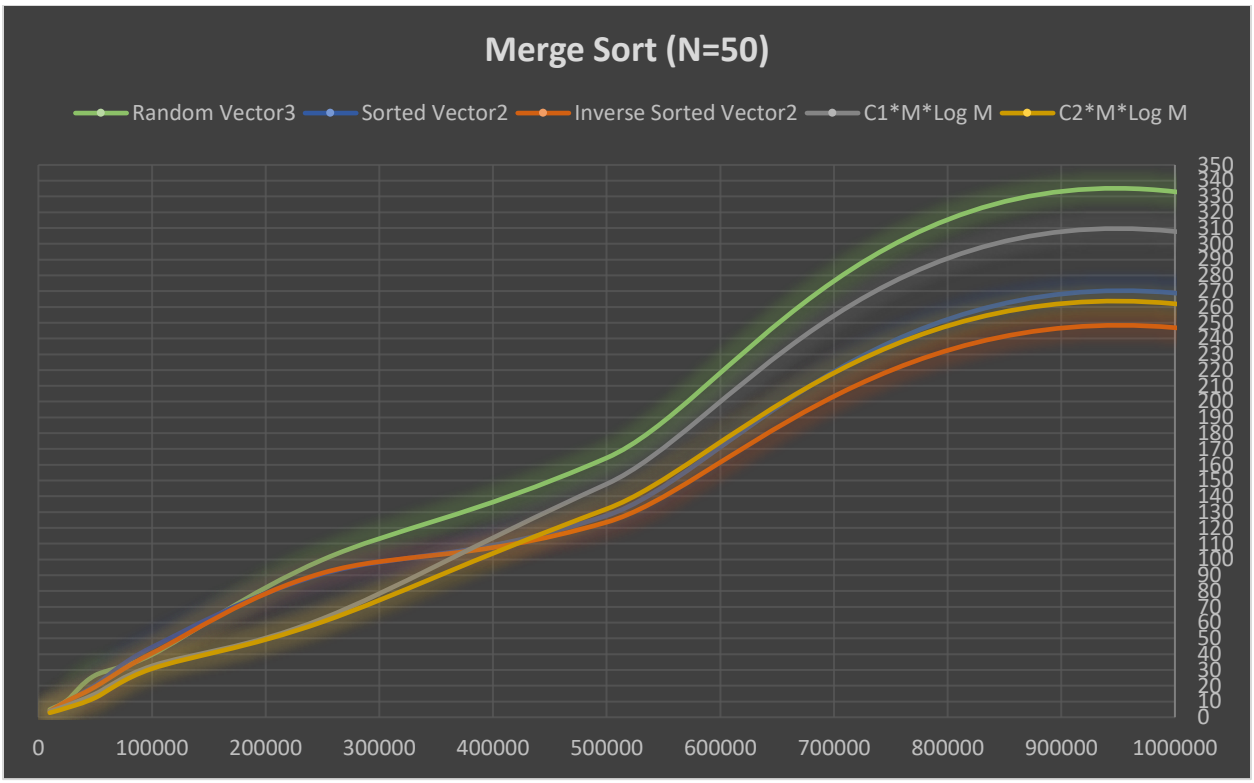
M	Random Vector	Sorted Vector	Inverse Sorted Vector
10000	4	2.83	2.67
25000	9.6	5.4	6
50000	19.2	16	11.6
100000	38.83	29.5	27.8
250000	98.2	72.8	70.93
500000	202.2	95.2	97.2
1000000	311.4	193.6	202.4
2500000	-	-	-



Finally, now lets verify the similar for the last case of the dimensions that is N= 50.

N=50

M	Random Vector	Sorted Vector	Inverse Sorted Vector
10000	4.85	3.5	3.84
25000	10.6	9.6	10.2
50000	26.6	20.4	19.3
100000	39.7	44.4	40.6
250000	99.8	90.8	91.4
500000	164.33	127.8	123.6
1000000	333	268.8	246.86
2500000	-	-	-



I can conclude that after running all the dimensions of N on various order of random integer vectors that Merge sort follows the time complexity of the Big O (N * Log N).