Q. 1 BubbleSort()

Algorithm analysis:

T(n) = O(n^2) for worst case
= O(n^2) for average case
= O(n) for best case

Input/Output:

Input is n = Number of element as in Input List
At, CPU: Dual Core 3Ghz, Memory: 4GB

1. For Random list

Input	**Time (in Sec)**
10	0.0
100	0.000094
500	0.001323
1000	0.005551
5000	0.100289
10000	0.392785
50000	9.724364
100000	**38.8822881**

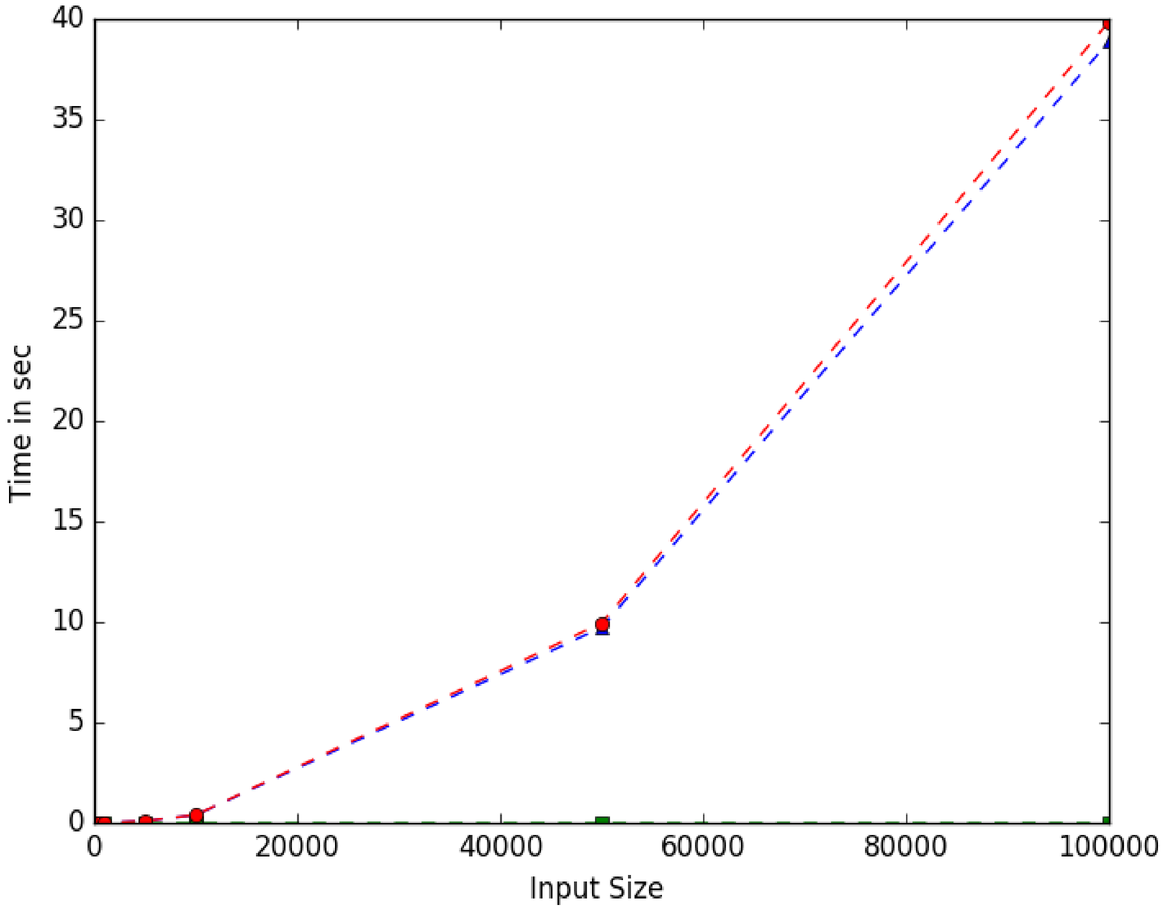
2. For Sorted list

Input	**Time (in Sec)**
10	0.000001
100	0.000002
500	0.000003
1000	0.000010
5000	0.000026
10000	**0.000087**
50000	**0.000424**
100000	**0.000714**

3. For Reverse Sorted list

Input	Time (in Sec)
10	0.000003
100	0.000098
500	0.002307
1000	0.006709
5000	0.103236
10000	**0.395791**
50000	**9.950184**
100000	**39.867644**

Complexity Graph:



Conclusion :
For Bubble sort it can be seen that **Worst case :** when list is reverse sorted, **Best case :** if list is sorted. In average case (Random list) it also take order of polynomial time for sorting.

Q. 2 SelectionSort()

Algorithm analysis:

T(n) = O(n^2) for worst case
= O(n^2) for average case
= O(n^2) for best case

Input/Output:

Input is n = Number of element as in Input List
At, CPU: Dual Core 3Ghz, Memory: 4GB

1. For Random list

Input	**Time (in Sec)**
10	0.000002
100	0.000028
500	0.000667
1000	0.001752
5000	0.045682
10000	**0.170458**
50000	**4.169171**
100000	**16.657978**

2. For Sorted list

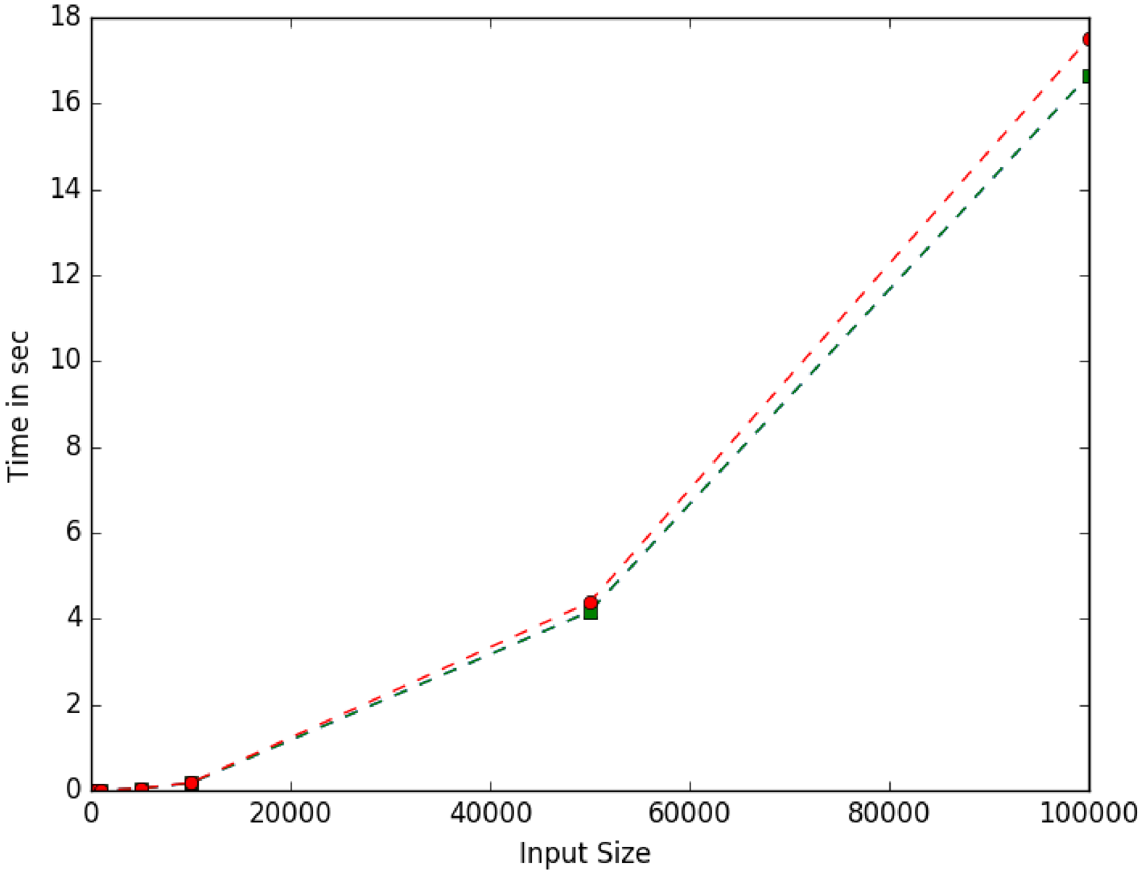
Input	**Time (in Sec)**
10	0.000001
100	0.000034
500	0.000521
1000	0.003108
5000	0.045903
10000	**0.169726**
50000	**4.167574**
100000	**16.662882**

3. For Reverse Sorted list

Input	**Time (in Sec)**
10	0.000002
100	0.000037
500	0.001110

1000	0.003384
5000	0.048474
10000	**0.177946**
50000	**4.386236**
100000	**17.514223**

Complexity Graph:



Conclusion :
For Selection sort **Worst case, Best case, Average Case** List does not matter as it does compare through out the list and hence for all the three cases are order of n^2 , It can also verified from the graph.

Q. 3 InsertionSort()

Algorithm analysis:

T(n) = O(n^2) for worst case
= O(n^2) for average case
= O(n) for best case

Input/Output:

Input is n = Number of element as in Input List
At, CPU: Dual Core 3Ghz, Memory: 4GB

1. For Random list

Input	**Time (in Sec)**
10	0.000002
100	0.000052
500	0.000899
1000	0.002850
5000	0.032414
10000	**0.112271**
50000	**2.800497**
100000	**11.202344**

2. For Sorted list

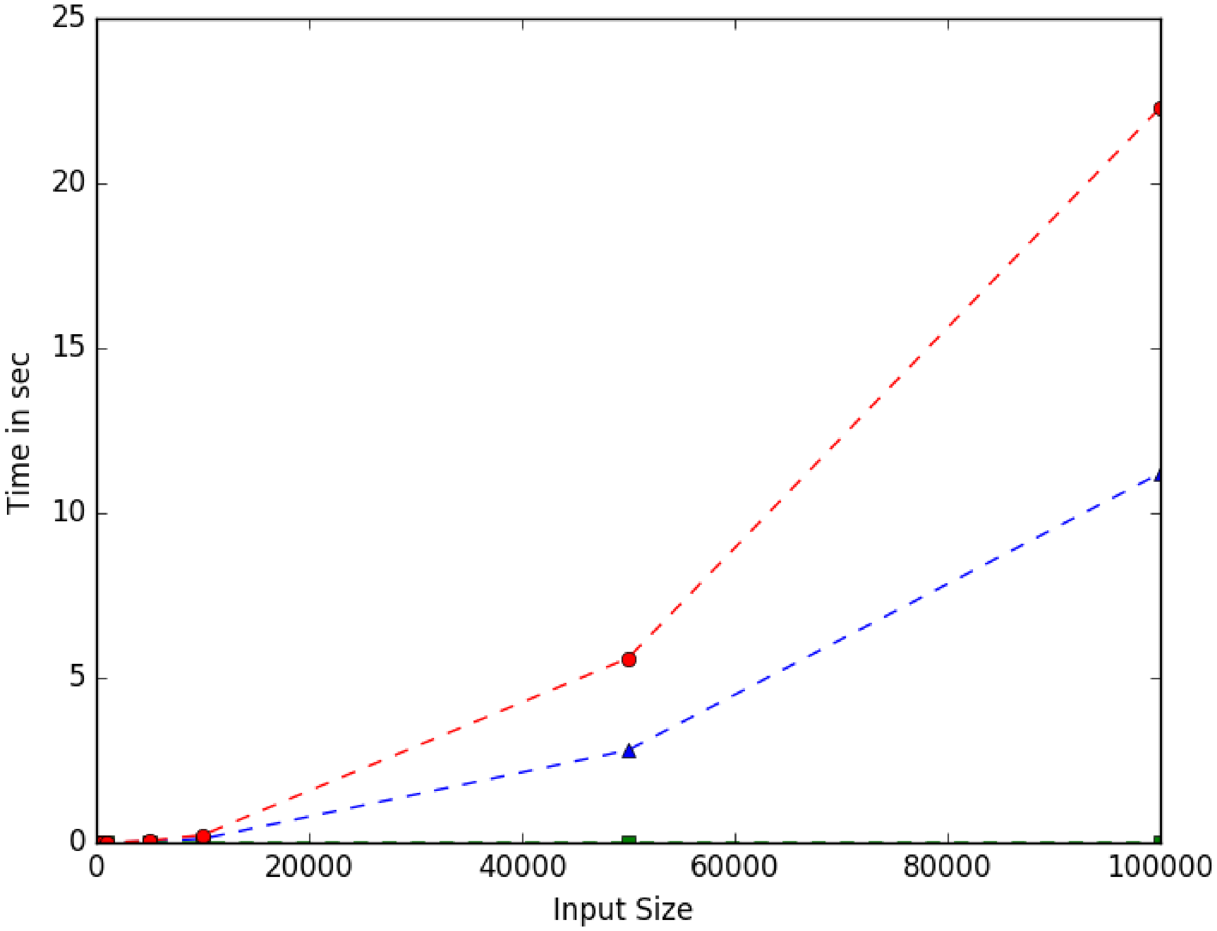
Input	**Time (in Sec)**
10	0.000001
100	0.000003
500	0.000008
1000	0.000007
5000	0.000051
10000	**0.000170**
50000	**0.000628**
100000	**0.001261**

3. For Reverse Sorted list

Input	**Time (in Sec)**
10	0.000002
100	0.000059
500	0.000547

1000	0.003310
5000	0.058911
10000	**0.224601**
50000	**5.591818**
100000	**22.295033**

Complexity Graph:



Conclusion :
For Insertion sort **Best case** list is sorted list as it run in linear time,
Worst case list is reverse sorted list as it took polynomial n^2 order of time.
Average Case List is random list in which time is less than n^2 but it is not linear it is some what faster average than other First 2 algorithms.

Q. 4 MergeSort()

Algorithm analysis:

T(n) = O(nLog(n)) for worst, average, best case

Input/Output:

Input is n = Number of element as in Input List
At, CPU: Dual Core 3Ghz, Memory: 4GB

1. For Random list

Input	**Time (in Sec)**
10	0.000004
100	0.000032
500	0.000176
1000	0.000499
5000	0.002137
10000	**0.002991**
50000	**0.014617**
100000	**0.030948**

2. For Sorted list

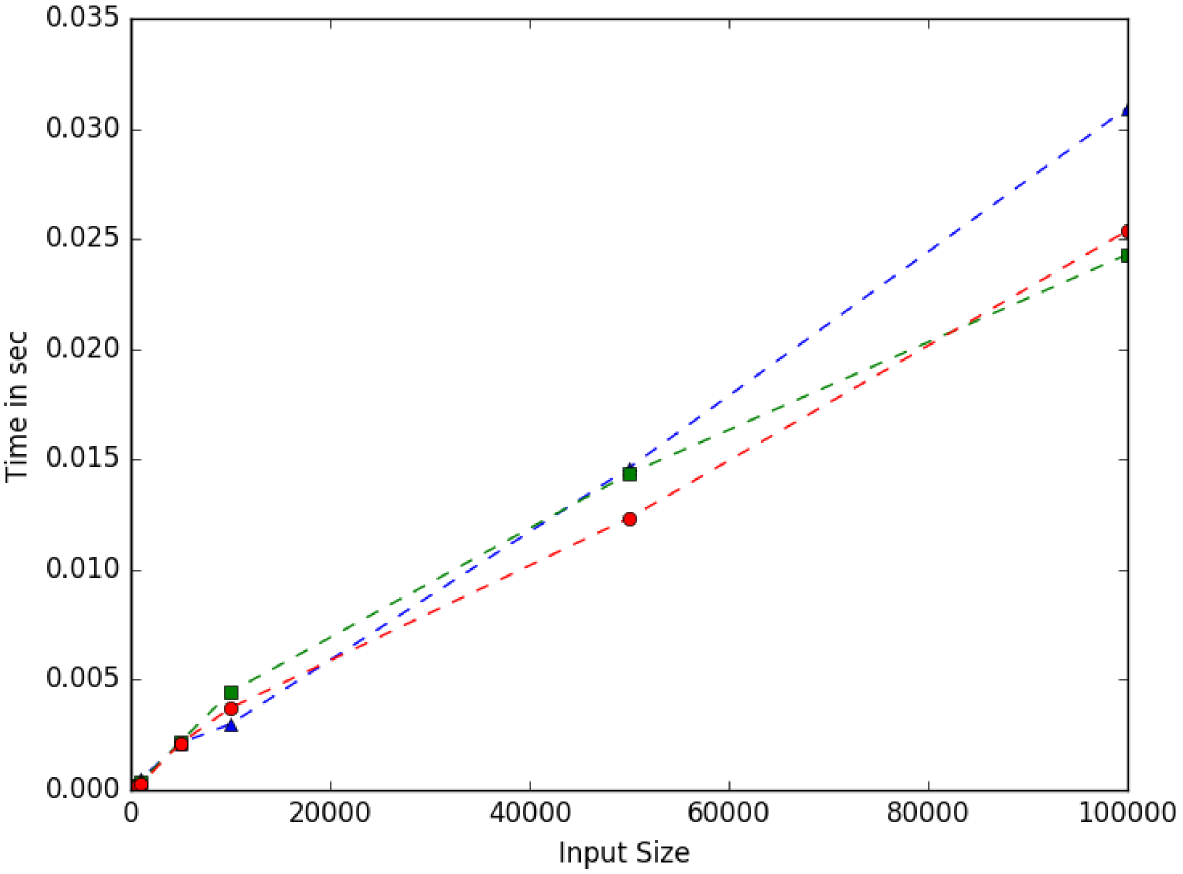
Input	**Time (in Sec)**
10	0.000004
100	0.000034
500	0.000187
1000	0.000394
5000	0.002189
10000	**0.004448**
50000	**0.014344**
100000	**0.024302**

3. For Reverse Sorted list

Input	**Time (in Sec)**
10	0.000004
100	0.000034
500	0.000133
1000	0.000282
5000	0.002134

10000	**0.003722**
50000	**0.012336**
100000	**0.025375**

Complexity Graph:



Conclusion :
For Merge sort **Best case** list, **Worst case** list, **Average Case** List It does not matter list is in which order it always run on nlogn time, It can also be verified on graph.

Q. 5 QuickSort()

Algorithm analysis:

T(n) = O(nLog(n)) for average
= O(n^2) for worst case
= O(n) for best case

Input/Output:

Input is n = Number of element as in Input List
At, CPU: Dual Core 3Ghz, Memory: 4GB

1. For Random list

Input	Time (in Sec)
10	0.000003
100	0.000016
500	0.000121
1000	0.000254
5000	0.001135
10000	**0.003234**
50000	**0.010411**
100000	**0.015513**

2. For Sorted list

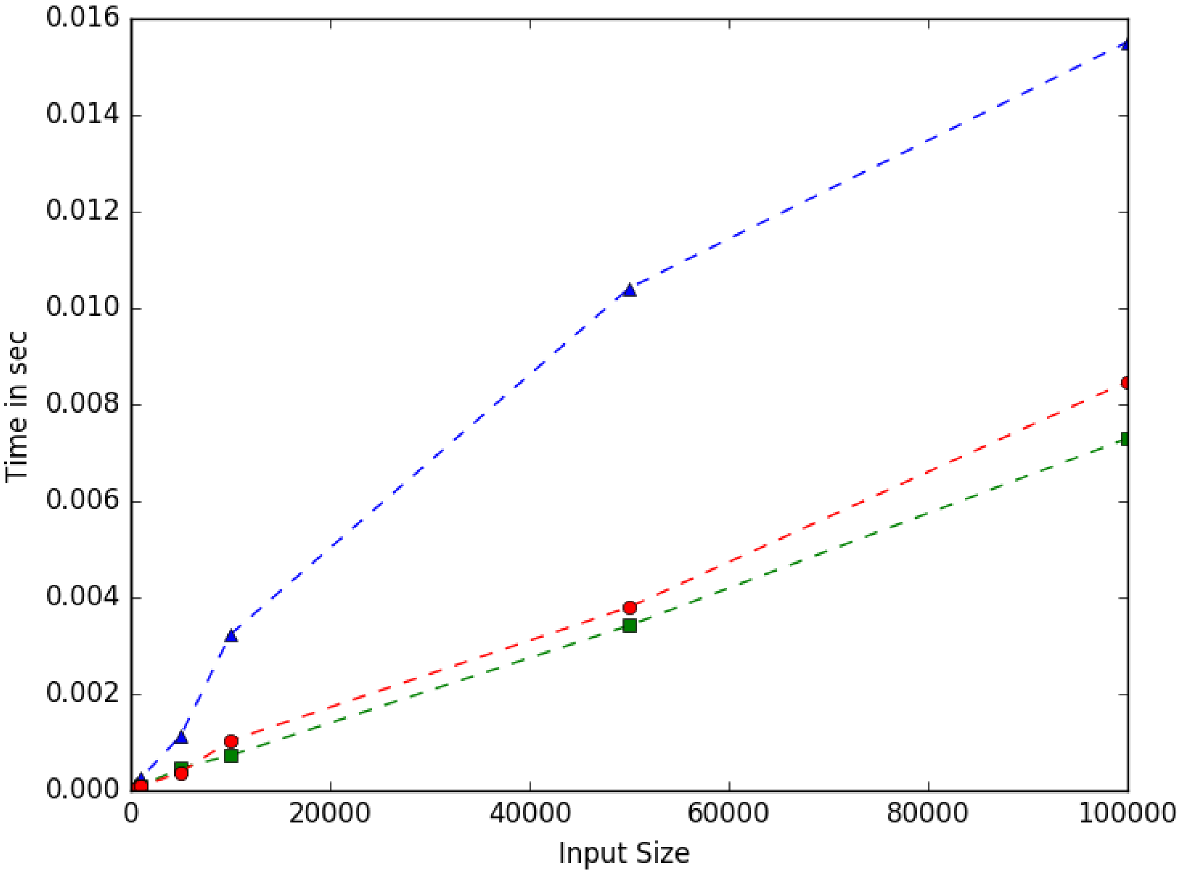
Input	Time (in Sec)
10	0.000002
100	0.000011
500	0.000037
1000	0.000086
5000	0.000463
10000	**0.000728**
50000	**0.003420**
100000	**0.007297**

3. For Reverse Sorted list

Input	Time (in Sec)
10	0.000003
100	0.000010
500	0.000025

1000	0.000086
5000	0.000374
10000	**0.001033**
50000	**0.003799**
100000	**0.008473**

Complexity Graph:



Conclusion :
For Quick sort the best case is that for which after each function call list devided into two equal half. And worst case if it devide in 1 vs n-1 list size.
As in curve for random list graph is nlogn and for both sorted and reverse sorted list it is of linear type this is because in both the case list is devided into two equal part due to sorted list as input.

Q. 6 CountingSort()

Algorithm analysis:

T(n) = O(n + k) where k is max size of elements to be in list

Input/Output:

Input is n = Number of element as in Input List
At, CPU: Dual Core 3Ghz, Memory: 4GB

1. For Random list

Input	**Time (in Sec)**
10	0.000004
100	0.000026
500	0.000112
1000	0.000337
5000	0.001455
10000	**0.001887**
50000	**0.008609**
100000	**0.017021**

2. For Sorted list

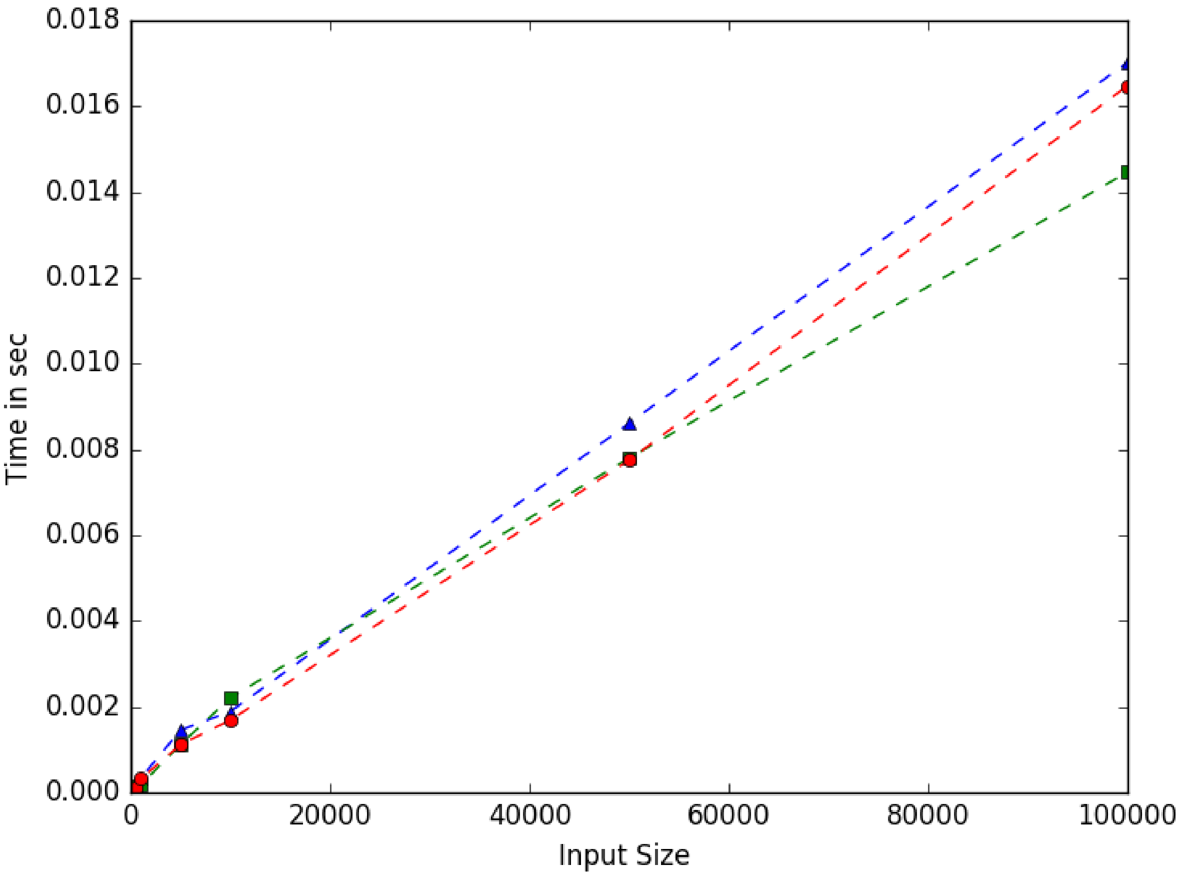
Input	**Time (in Sec)**
10	0.000005
100	0.000025
500	0.000149
1000	0.000180
5000	0.001119
10000	**0.002208**
50000	**0.007797**
100000	**0.014461**

3. For Reverse Sorted list

Input	**Time (in Sec)**
10	0.000003
100	0.000026
500	0.000113
1000	0.000319

5000	0.001119
10000	**0.001689**
50000	**0.007746**
100000	**0.016479**

Complexity Graph:



Conclusion :
For counting sort as expected it can also be verified that the graph is linear, since it depends more on the size of count list used to sort the element, and hence for all three list it does not depend upon list order for Best Worst or Average case.

Q. 7 HeapSort()

Algorithm analysis:

T(n) = O(nlogn) for every case

Input/Output: Input is n = Number of element as in Input List
At, CPU: Dual Core 3Ghz, Memory: 4GB

1. For Random list

Input	**Time (in Sec)**
10	0.000002
100	0.000026
500	0.000215
1000	0.000189
5000	0.002932
10000	**0.004771**
50000	**0.017548**
100000	**0.035945**

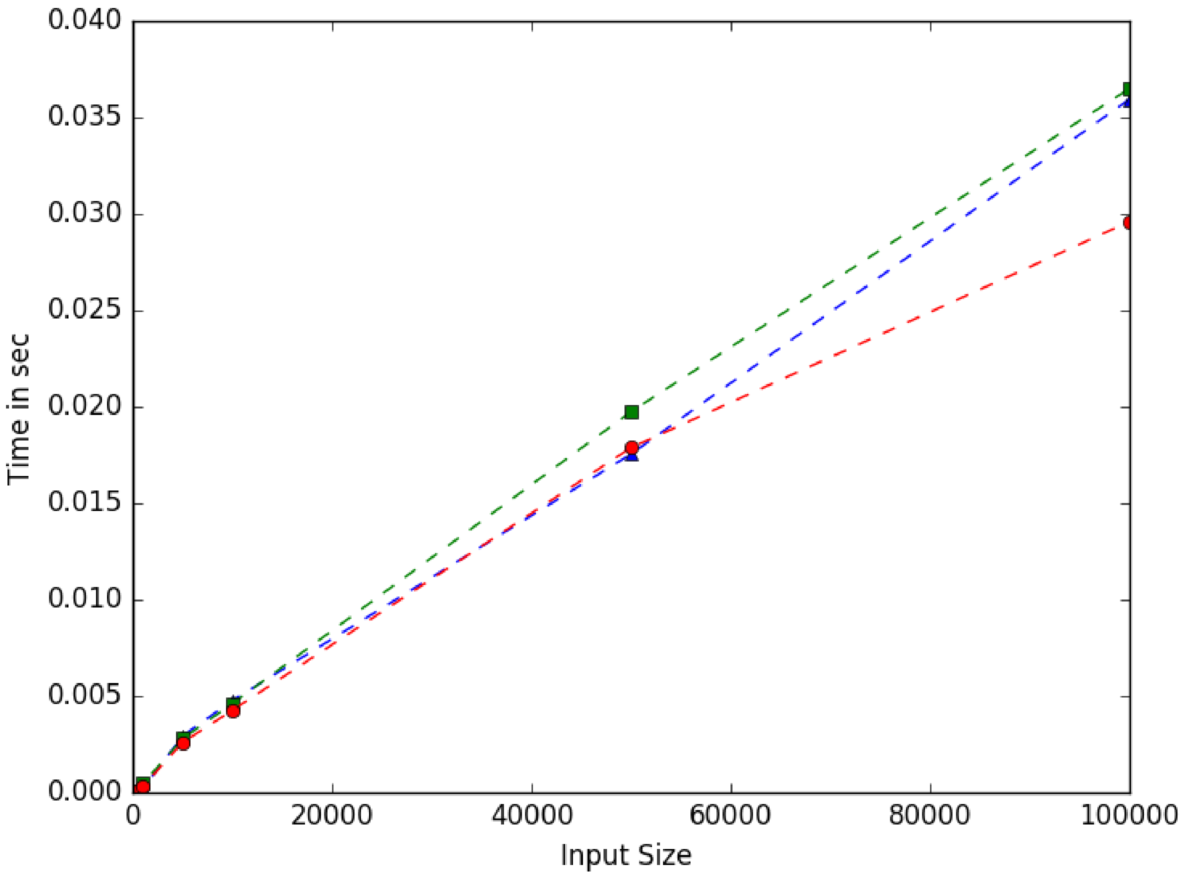
2. For Sorted list

Input	**Time (in Sec)**
10	0.000004
100	0.000017
500	0.000105
1000	0.000460
5000	0.002793
10000	**0.004613**
50000	**0.019744**
100000	**0.036496**

3. For Reverse Sorted list

Input	Time (in Sec)
10	0.000003
100	0.000030
500	0.000111
1000	0.000307
5000	0.002597
10000	**0.004264**
50000	**0.017881**
100000	**0.029594**

Complexity Graph:



Conclusion :
In Heap sort all three complexity are nlogn, it can also be verified from graph hence there is no perticular best, worst list for it.

Q. 8 RadixSort()

Algorithm analysis:

T(n) = O(nw) where w is word size of size of digits in max num

Input/Output:

Input is n = Number of element as in Input List
At, CPU: Dual Core 3Ghz, Memory: 4GB

1. For Random list

Input	Time (in Sec)
10	0.000006
100	0.000020
500	0.000092
1000	0.000311
5000	0.000773
10000	**0.003906**
50000	**0.012594**
100000	**0.019010**

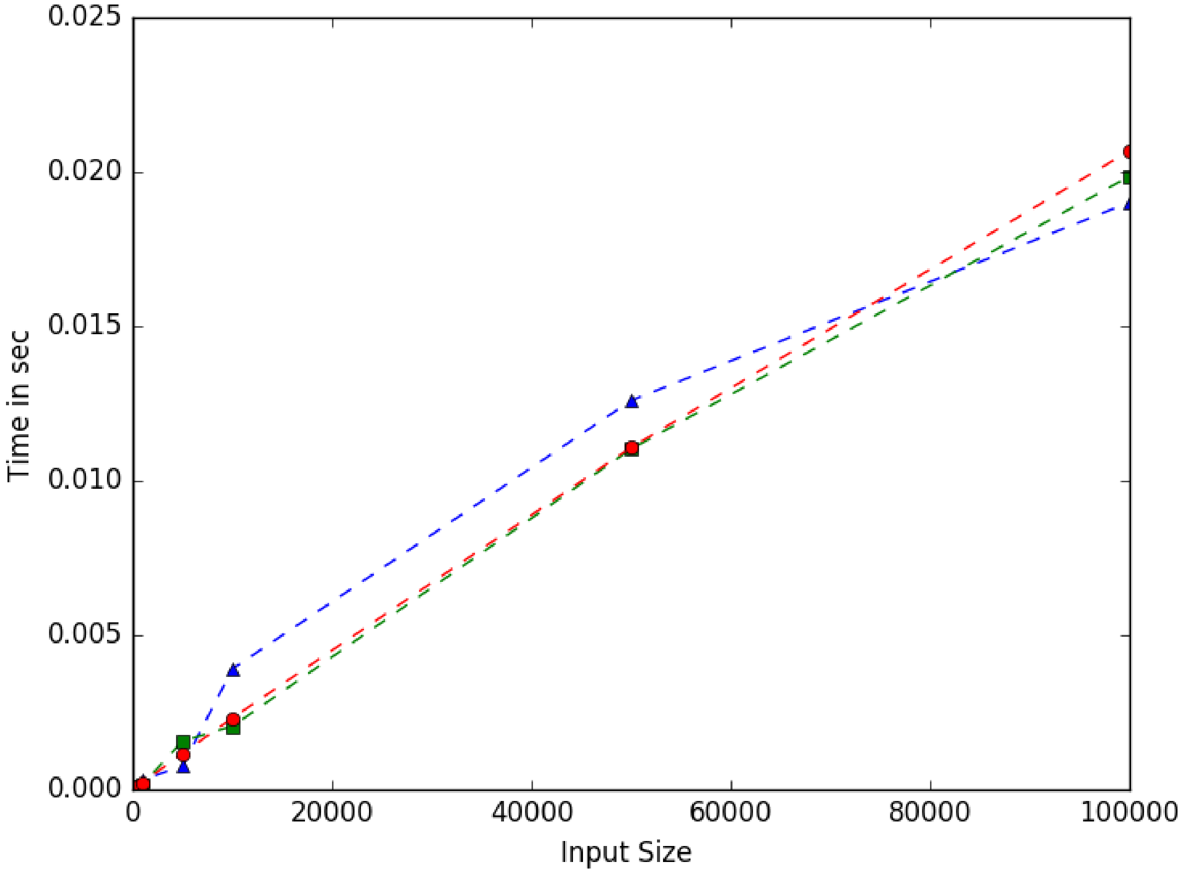
2. For Sorted list

Input	Time (in Sec)
10	0.000003
100	0.000014
500	0.000118
1000	0.000175
5000	0.001570
10000	**0.002047**
50000	**0.011025**
100000	**0.019851**

3. For Reverse Sorted list

Input	Time (in Sec)
10	0.000004
100	0.000028
500	0.000125
1000	0.000233
5000	0.001167
10000	**0.002316**
50000	**0.011080**
100000	**0.020677**

Complexity Graph:



Conclusion :
In Heap sort all three complexity are nlogn, it can also be verified from graph hence there is no perticular best, worst list for it.