Performance of Sorting Algorithms

Akhilesh Godi, CS10B037

Introduction

The main motive of the experiment was to compare the implementations of 3 sorting algorithms namely - Inserion Sort, Bubble Sort and Merge Sort. The size of input was varied in powers of 10 and the performance of each of the algorithms was quantitatively measured by calculating the time taken for the list to get sorted using the in-built libraries. A list with random numbers is first generated, and the same list is used for analyziing the performance of all the three algorithms. The performance of the algorithm was also tested using two different Data Structure (List implementations) – Array Lists and Linked Lists. The use of Generics in the code makes the code universal, by sorting lists of any fixed Data type. The performance of the algorithms is also tested for strings, as was done for numbers to check if using a different data type has an influence on it's performance.

In general, one would expect that the performance of Mergesort [Average case : O(n*log(n)] would be better in comparison to Bubble Sort or Insertion Sort [Average Case : O(n^2)]. One also expects that strings would take larger time for sorting

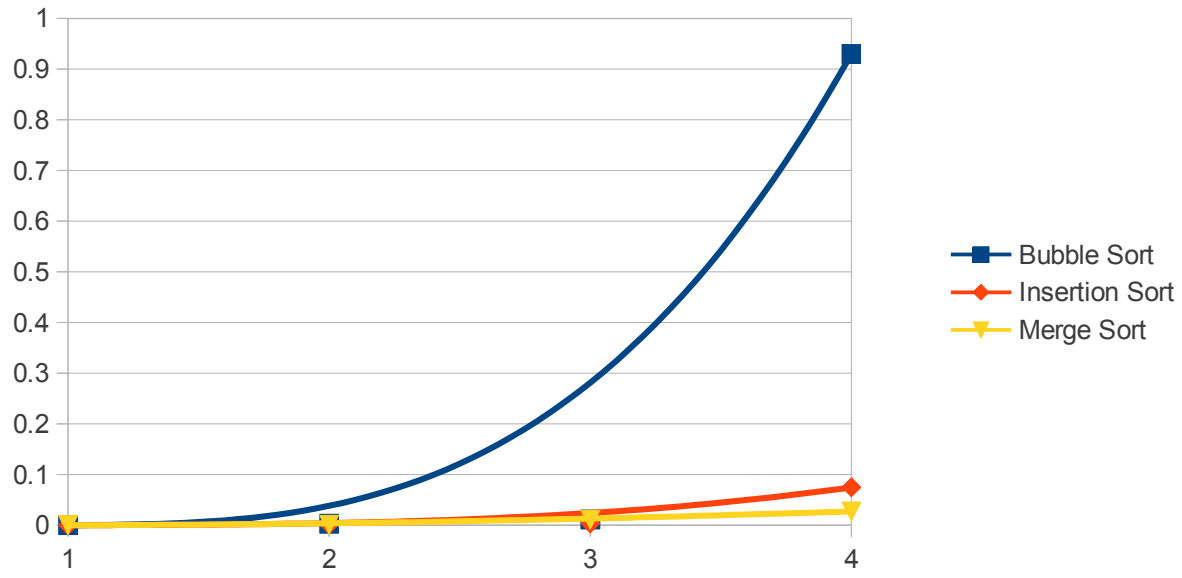
Algorithm Implementations

The standard algorithms were implemented and tested for their performance. Some small modifications were made after checking out the experimental results. It is seen that insertion sort works better for smaller input. Instead of recursively calling mergesort on the list, we call Insertion sort when the list size is 10 or less, in anticipation that the performance would be better.

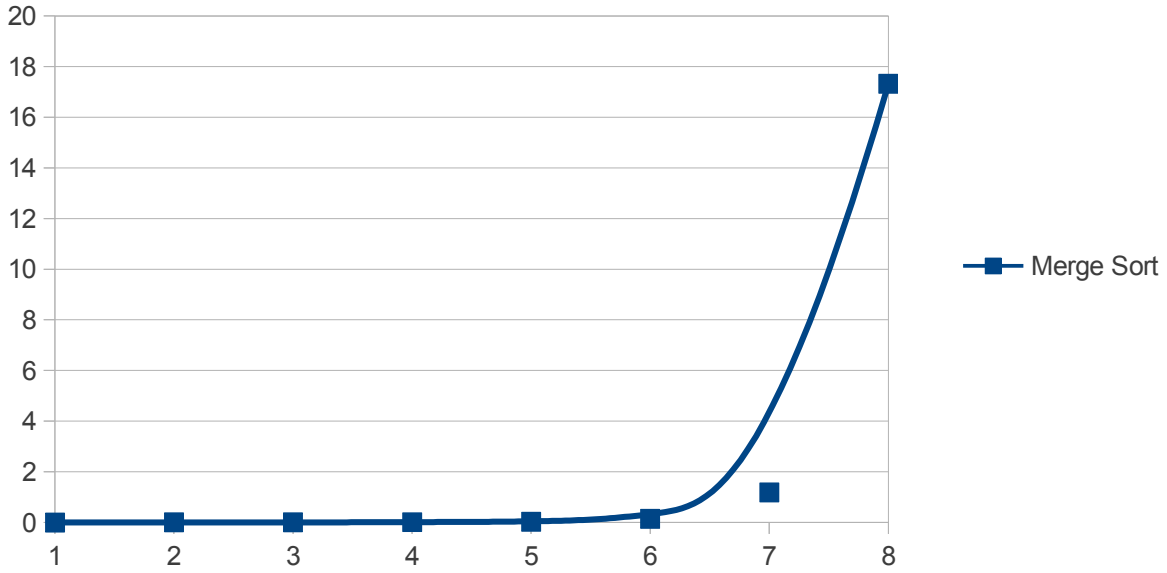
Experimental Results

For integers :

Number of Elements	ArrayList			LinkedList		
	Bubble Sort	Insertion Sort	Merge Sort	Bubble Sort	Insertion Sort	Merge Sort
10e2	0.0029	0.0012	0.0009	0.0053	0.0111	0.0018
10e3	0.0113	0.0037	0.0108	1.2413	0.3677	0.0131
10e4	0.9298	0.0748	0.0273	>>5mins	429.613	0.2489
10e5	94.5464	10.9104	0.1177	----	>5mins	41.5968
10e6	>5mins	>5mins	1.0973	----	----	>5mins
10e7	----	----	17.3194	----	----	----



Above is a plot for Sorting integers of various size. [X-Axis - log(Input Size) ; Y-Axis – Time in seconds]



Above is a plot for the array implementation of Merge Sort (almost in agreement with the expected n(logn) behaviour.)

For Strings:

Number of Elements	ArrayList			LinkedList		
	Bubble Sort	Insertion Sort	Merge Sort	Bubble Sort	Insertion Sort	Merge Sort
10e2	0.0030	0.0013	0.0016	0.0050	0.0024	0.0019
10e3	0.0269	0.0054	0.0109	1.6763	0.4774	0.0118
10e4	3.0525	0.3062	0.1515	>5mins	>5mins	0.5219
10e5	>5mins	85.7999	16.8523	----	----	111.947

10e6	----	>5mins	>5mins	----	----	>5mins
10e7	----	----	----	----	----	----

Conclusions

Interestingly it was observed that Insertion sort is more often efficient than Bubble sort, though they are of the same class algorithmically. As expected, the performance of the algorithms reduced considerably when we used Lists of Strings instead of integers. As the input size is increased, it is observed that the average time taken to sort the input increased and in most cases did comply to the expected orders. However, considerable deviations were observed. As expected, the performance was better in the case of Array Lists in comparison to that of Linked Lists and is justified as one does not have access to every element in the case of a Linked List and has to traverse the list. As can be seen from the tabulations, Insertion sort worked better with an Array List in comparison to that of Merge Sort with a Linked List.

`Insertion sort took  0.000032 seconds for an input size of 10 and Merge sort took  0.000096 seconds for an input size of 10.`

So, taking the above fact into consideration, an attempt was made to implement the slightly modified version of the sort algorithm, as discussed previously. This modification resulted in considerable increase in performance. Merge Sort took `17.3194 seconds` for sorting an Array List of size 10^7 integers, where as our modified sorting algorithm took `15.6977 seconds` for sorting the same array list. Thus it can be concluded that for very large input size our modified sort algorithm practically works faster than the Merge Sort,