Algorithmic Development

Today your mission is to develop a Java class that implements several elementary (and silly) sorting algorithms. The problem we want our algorithms to solve is sort an input array of integers into ascending order and output the resulting array.

Possible steps to follow

1. Create a new java class
2. Implement the following sorting algorithms as public static functions within your class that take an array of integers and sorts the array, outputting a sorted array of integers:
    a. Selection sort
    b. Insertion Sort
    c. A silly sort (either from the list below or of your own making)
3. Create a simple framework for generating input arrays of various sizes (e.g., 10, 1000, 100,000) and then testing the performance over several runs
4. Print the resulting sorted array: Implement a function to print out all elements in the array
5. Time the performance of the previous step on your 3 algorithms and output the execution times for various input sizes (e.g. 10,100,1000) on a graph
6. Justify the results of your experiments for the algorithms by proposing the algorithm complexity in big-O notation
7. BONUS: adjust your insertion sort algorithm to be an unstable sort

1.

This class is called Sort.java and is available in the github repository.

2.

see Sort.java

3.

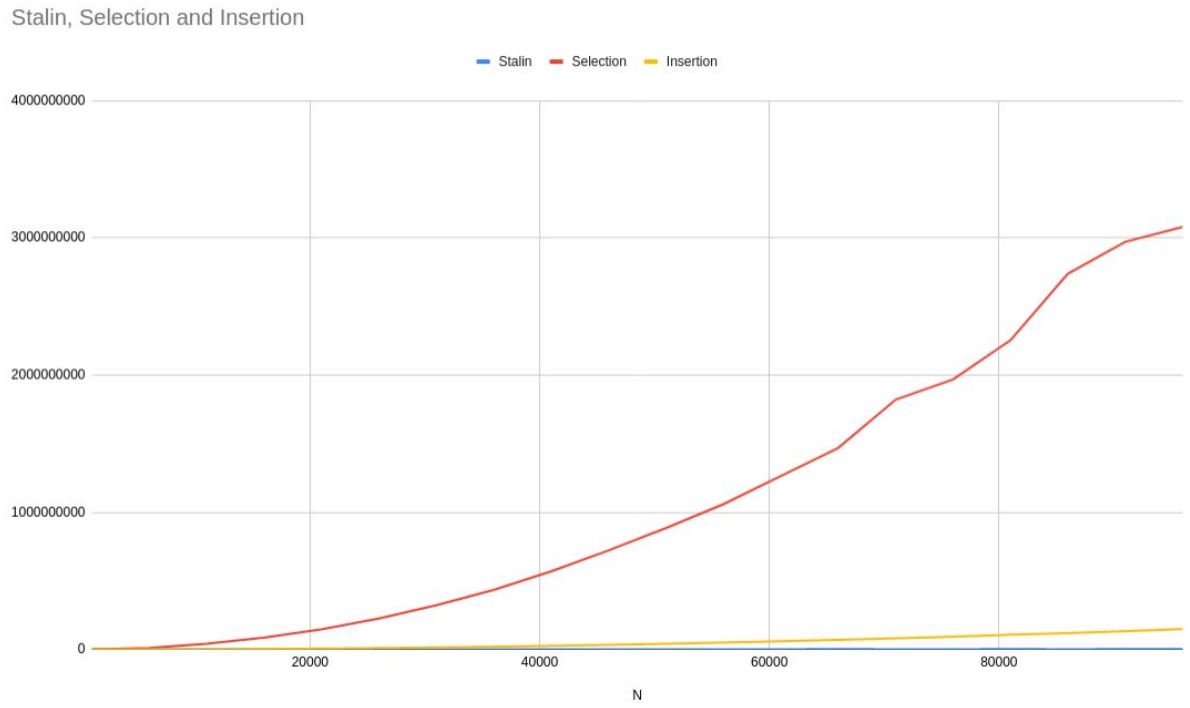
see Sort.randomArray()

4.

I used Arrays.toString()

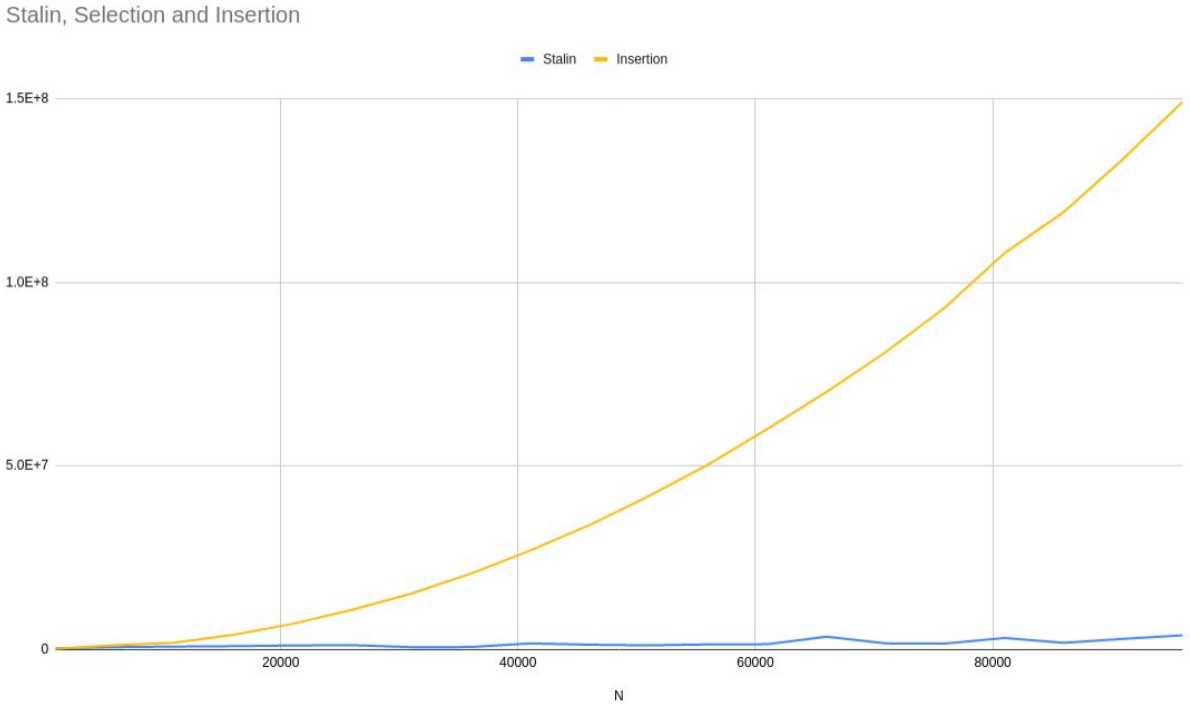
5.

Please find below a graph comparing the running time of selection, insertion, and stalin sort for various inputs



As can be seen from the graph above, stalin sort is significantly faster than either insertion or selection sort. There is, howver, a slight surprise in that insertion sort and stalin sort almost look comparable on the graph. This, however is a result of the skewed axes of the graph, as a result of the terrible performance of this implementation of selection sort.

Below, please find a graph of insertion sort vs stalin sort, which better highlights the growth rate of this implementation of insertion sort.



Clearly, we can see from the above graph that insertion sort's growth rate is definitely not comparable to that of stalin sort.

6.

The experimental results showed that selection sort was slower than insertion sort, which was again slower than selection sort. Since selection sort and insertion sort are $O(n^2)$, it is not surprising that they are both the slowest algorithms. While the difference between selection sort and insertion sort was measurable, this does not undermine the fact that their growth rates are similar. Big-Oh notation concerns asymptotic analysis, meaning that we are looking for the upper bound in terms of the **growth rate** of the algorithms. Two algorithms with equivalent Big-Oh classifications will not always perform identically. In fact, the opposite is almost always the case.

Stalin sort being much faster than the two others is no surpise, as it is an O(n) algorithm, which is almost as fast as an algorithm can be, slower only in terms of growth rate than O(log n) and O(1) functions.

7.

Relatively simple alteration. In cases of equality, move the item in the unsorted array to the position currently occupied by its equal counterpart in the sorted array, as opposed to the position immediately after it.