Pawan Pinjarkar
CSCI B505 Fall 2017
Programming Assignment 1
ppinjark@umail.iu.edu

Below are the graphs demonstrating performance comparison of bubble sort and insertion sort.

Case 1: Random Input

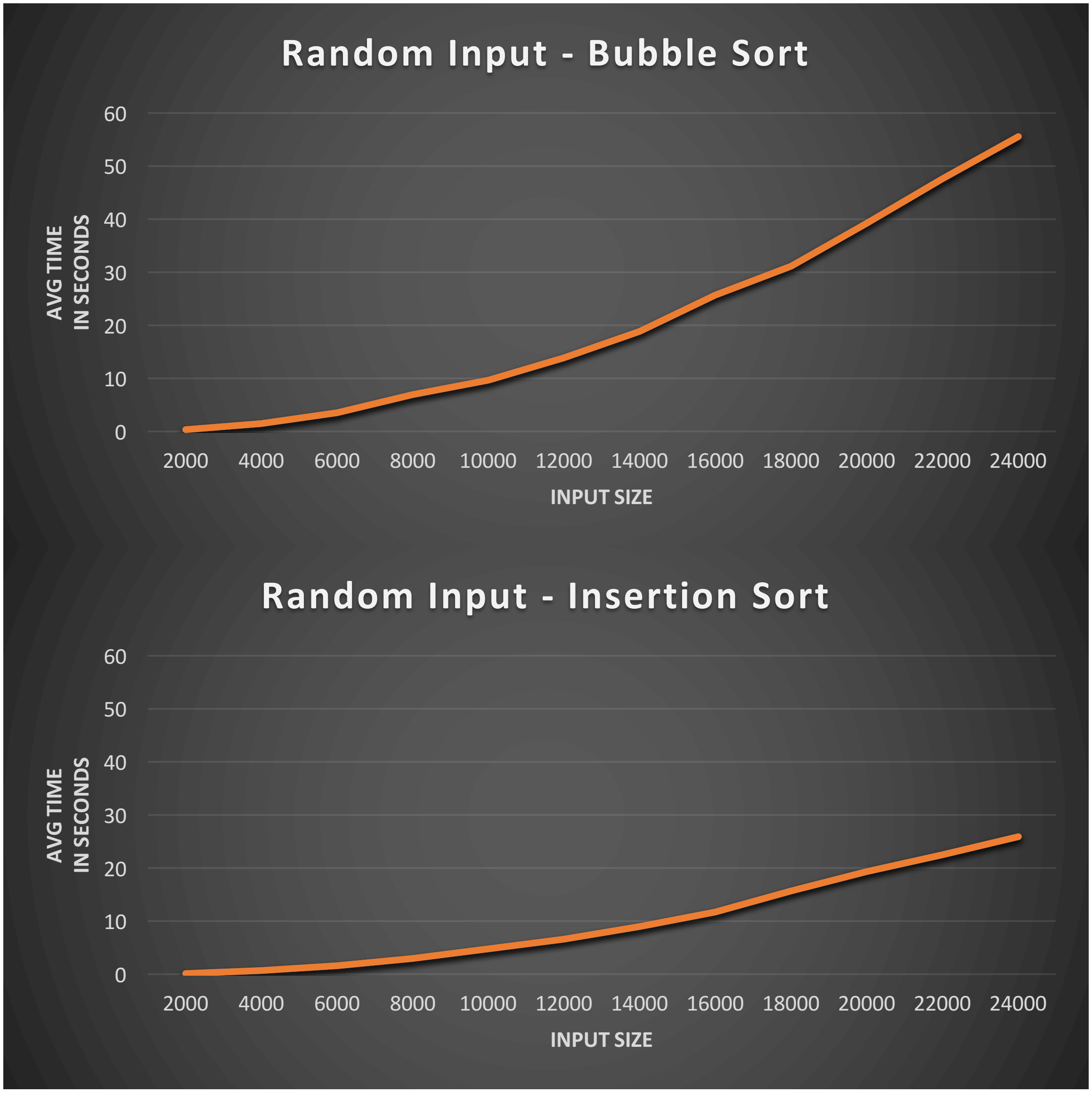

Fig. 1

Table 1 shows the data used for plotting the above graph in Fig. 1 –

Bubble sort		Insertion sort	
Input size	**Avg. time in seconds**	**Input size**	**Avg. time in seconds**
2000	0.416873854	2000	0.183502007
4000	1.532676508	4000	0.75148754
6000	3.502779205	6000	1.660017608
8000	6.96896785	8000	2.951377846
10000	9.656003242	10000	4.815588106
12000	13.88529165	12000	6.646351303
14000	18.87892097	14000	8.989430204
16000	25.73515911	16000	11.72140948
18000	31.15335715	18000	15.66344959
20000	39.27585936	20000	19.34623191
22000	47.59274868	22000	22.53599155
24000	55.52955357	24000	25.9677769

Table 1

Case 2: Non-decreasing Input

This is a **best-case input** as the input list/array is already sorted.

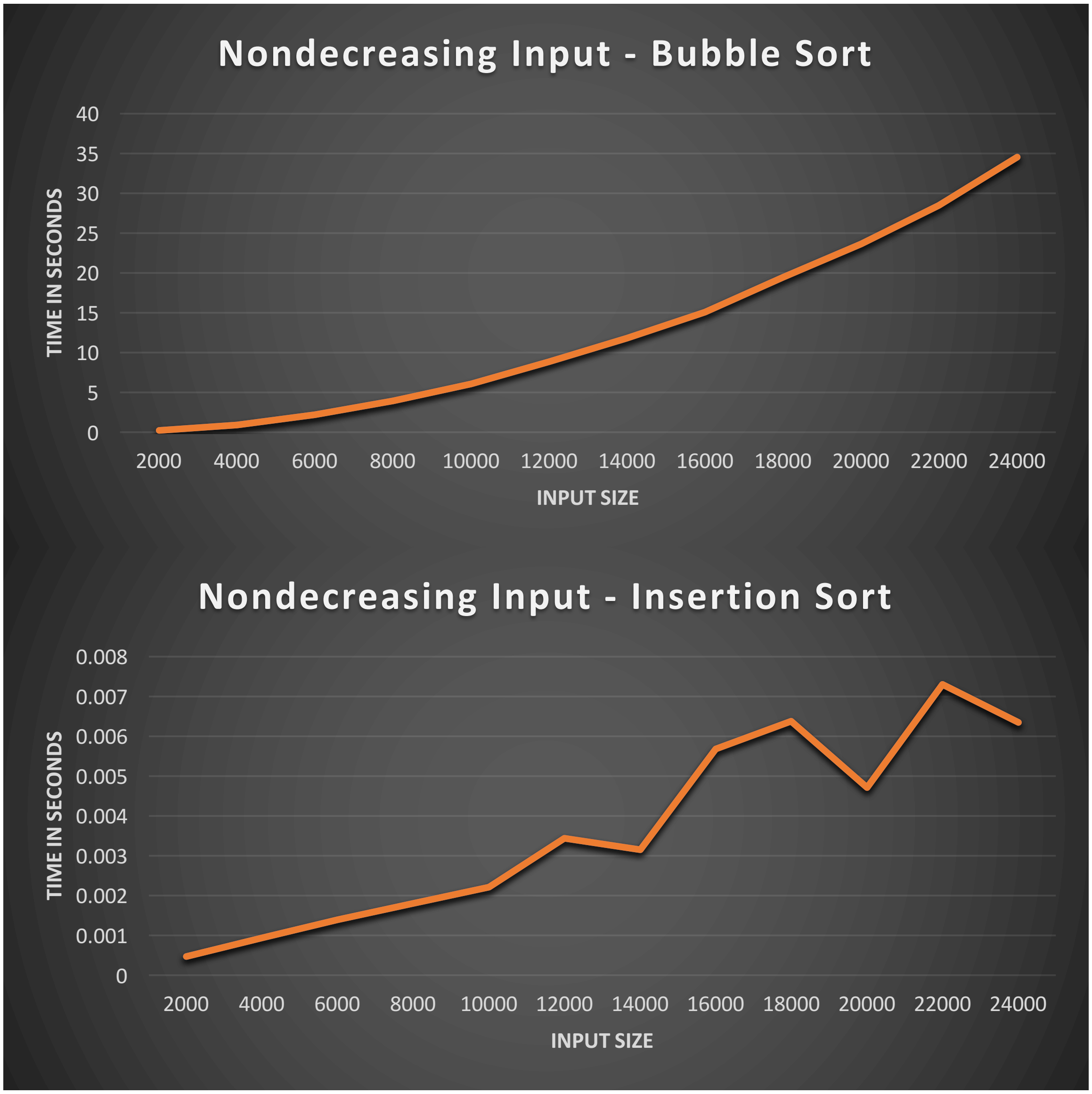

Fig. 2

Note: Please refer Fig.4 for linear graph of insertion sort which has the same data with the Y axis plotted from 1 to 10.

Table 2 shows the data used for plotting the above graph in Fig. 2 –

Bubble sort		Insertion sort	
Input size	**Avg. time in seconds**	**Input size**	**Avg. time in seconds**
2000	0.235520984	2000	0.000468524
4000	0.921095763	4000	0.00093173
6000	2.189044903	6000	0.001386806
8000	3.93979906	8000	0.001805099
10000	6.04015417	10000	0.002209174
12000	8.84885108	12000	0.003428362
14000	11.77036708	14000	0.003141664
16000	15.0174219	16000	0.005674783
18000	19.43386721	18000	0.006368821
20000	23.60454727	20000	0.00471346
22000	28.42815311	22000	0.007296222
24000	34.53163816	24000	0.006347004

Table 2

Case 3: Non-increasing Input

This is a **worst-case input** as the input list/array is sorted in reverse order.

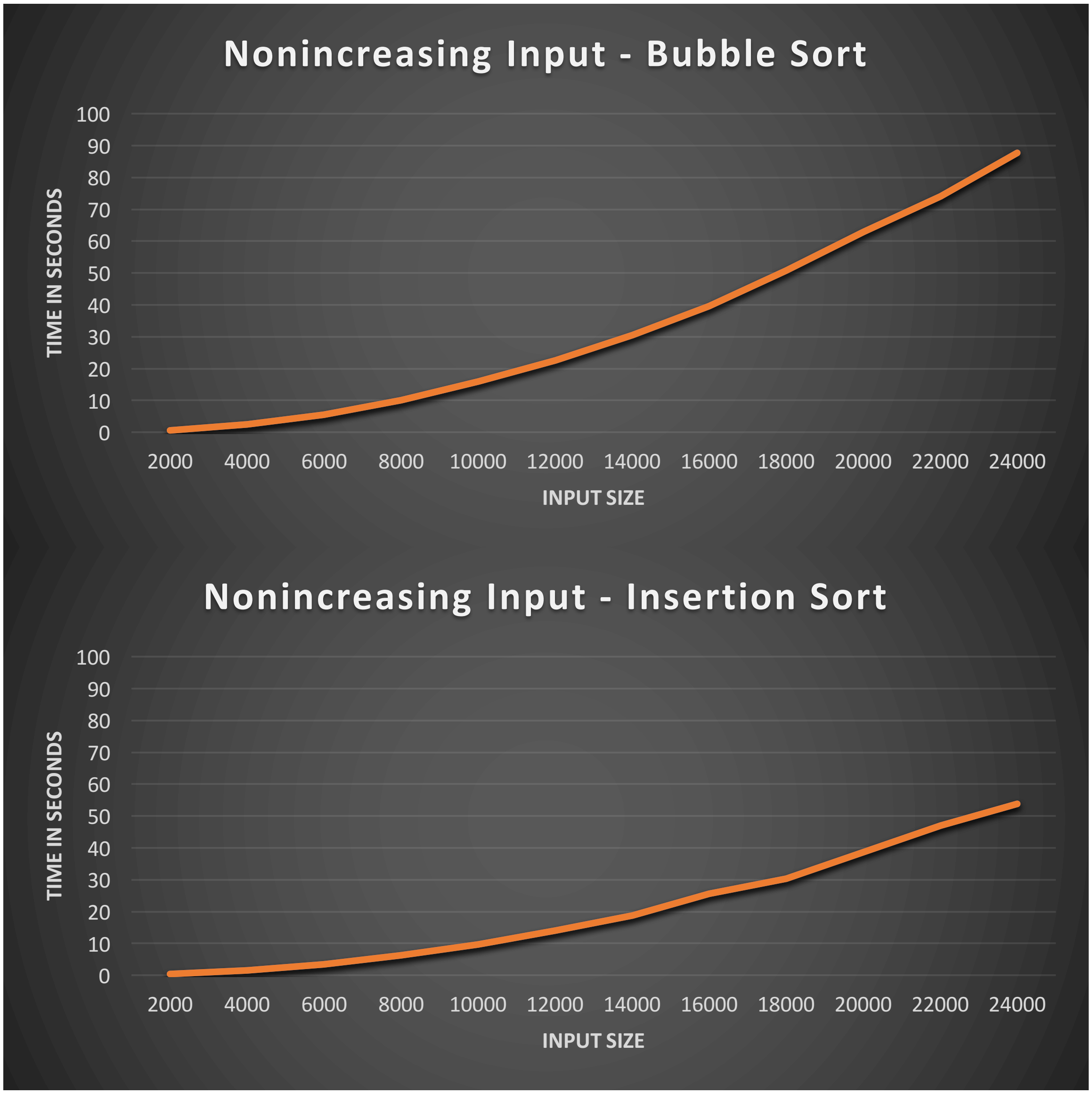

Fig. 3

Table 3 shows the data used for plotting the above graph in Fig. 3 –

Bubble sort		Insertion sort	
Input size	**Avg. time in seconds**	**Input size**	**Avg. time in seconds**
2000	0.582553692	2000	0.373676463
4000	2.509741997	4000	1.556398307
6000	5.63972278	6000	3.440014391
8000	10.14383619	8000	6.368040002
10000	16.03050036	10000	9.685945009
12000	22.63728406	12000	14.14480404
14000	30.4971381	14000	18.83060271
16000	39.64956573	16000	25.64387328
18000	50.78341408	18000	30.41777429
20000	62.85162209	20000	38.58485734
22000	74.02564314	22000	47.04039731
24000	87.65771879	24000	53.75836705

Language used for coding: Python
Why did I choose Python?
I am basically a java developer and have an extensive experience in java and JavaScript languages. Python is a very powerful programming language and is heavily used in data science field such as data mining, artificial intelligence, machine learning etc. I believe Python helps the developer to do more in less lines of code. I used Python over java because I wanted to have a hands-on experience in Python. Python programs are generally expected to run slower than Java programs, but they also take much less time to develop.

I believe if we make use of Python correctly and efficiently, Python program can also give better performance. The python modules make the code very easy to maintain and also has a rich support of third party libraries. I had learnt Python as a subject in Spring 17 semester and want to do more and more python coding as its faster to develop Python programs.

Tools/Platform used for plot: Microsoft Excel
Why did I choose Microsoft Excel?
I could have used Python module – matplotlib, to plot the graphs. However, I choose Microsoft Excel as its super easy to plot the graphs and it provides rich features to get the visually appealing plots. Moreover., as I am working full time and also doing my online MS, Excel

helped me save the time to plot the graphs which I could have done with Python, if it has been a mandatory thing.

How did I create/store the data?

To generate the data with random integers, I wrote a small python program which used Python's **random** module. I created separate files with a python program. The program generated the random integers and also saved the list into the separate files. I saved the data in normal .txt files.

For the non-decreasing input, I took one file from each varying input size (2000, 4000, 6000, …, 24,000) of data set from the first part i.e. random input size case. I then read the file with a separate python program into a list and then sorted the list as below.

sortedNumberList = sorted(numberList)

Then saved the sorted list in the non-decreasing format into a file using Python's **simplejson** module. I then manually removed the square brackets and comma which were added in the data set while saving the list to file using simplejson python module.

For non-increasing input, I followed the same approach as mentioned above but while sorting the list, I reverse sorted the list as below

sortedNumberList = sorted(numberList,reverse=True)

Did I face any difficulties?

I faced couple of issues at first understanding the requirement related to the input to be used for 3 cases as mentioned in the assignment. During coding, I faced some issues while reading files, converting the data from file into a python list and then sorting it. I had to make use of try/catch exception handling to overcome the issue. Apart from this, I did not really face any issues.

Conclusion:

Overall insertion sort is better at performance i.e. running time when compared to bubble sort. Bubble sort has worst-case and average complexity both as O(n^2) while insertion sort has O(n) as the best-case time complexity, O(n^2) as the worst-case time complexity. In best case, insertion sort has a linear running time as the input was already sorted and it was very . Please check below graph in Fig. 4 which represent the best-case scenario.

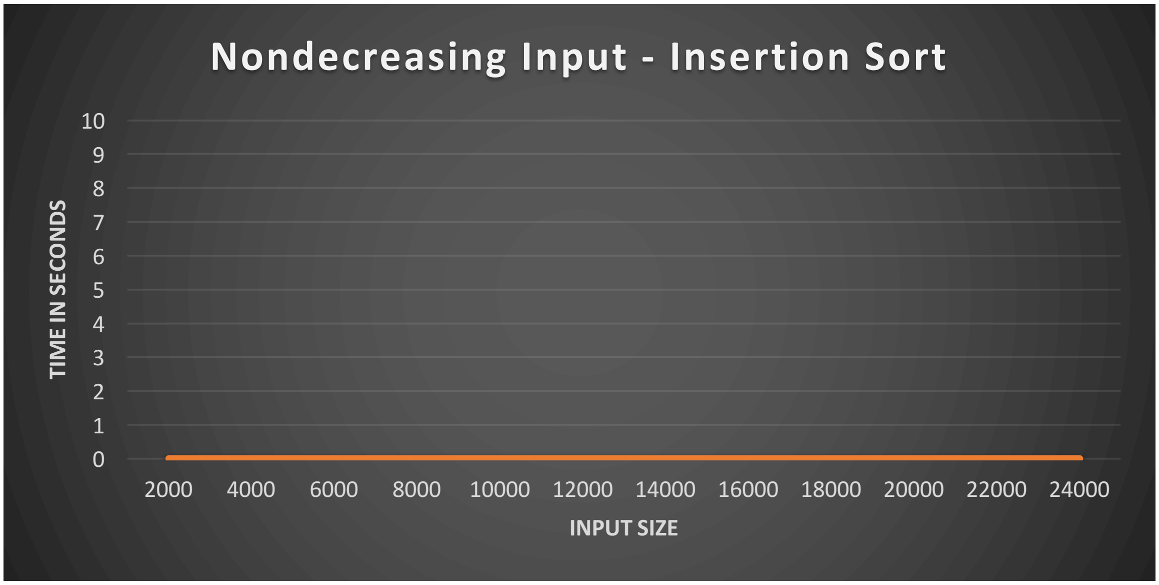

Fig.4

The Fig.1 shows that with random input data, both the algorithms have quadratic running time for case 1. For case 2 i.e. with non-decreasing order, bubble sort has the quadratic running time while insertion sort (also refer Fig. 4) has the linear running time.

When compared the Fig.3 with Fig.1 and Fig.2, it is clear that bubble sort does take more time than insertion sort. In worst-case, time complexity of bubble sort is O(n^2) which is same as insertion sort.

I observed quadratic function in all the plots except in Fig. 2 and Fig. 4 which is a linear function.

I don’t think bubble sort is really used in real life applications because of its inefficient time complexity. It is easy to learn and can be beneficial for a very small problem size. Insertion sort is used in standard C++ sort function. Insertion sort is more efficient in practice on small array/lists and lists which are already processed by another algorithm.