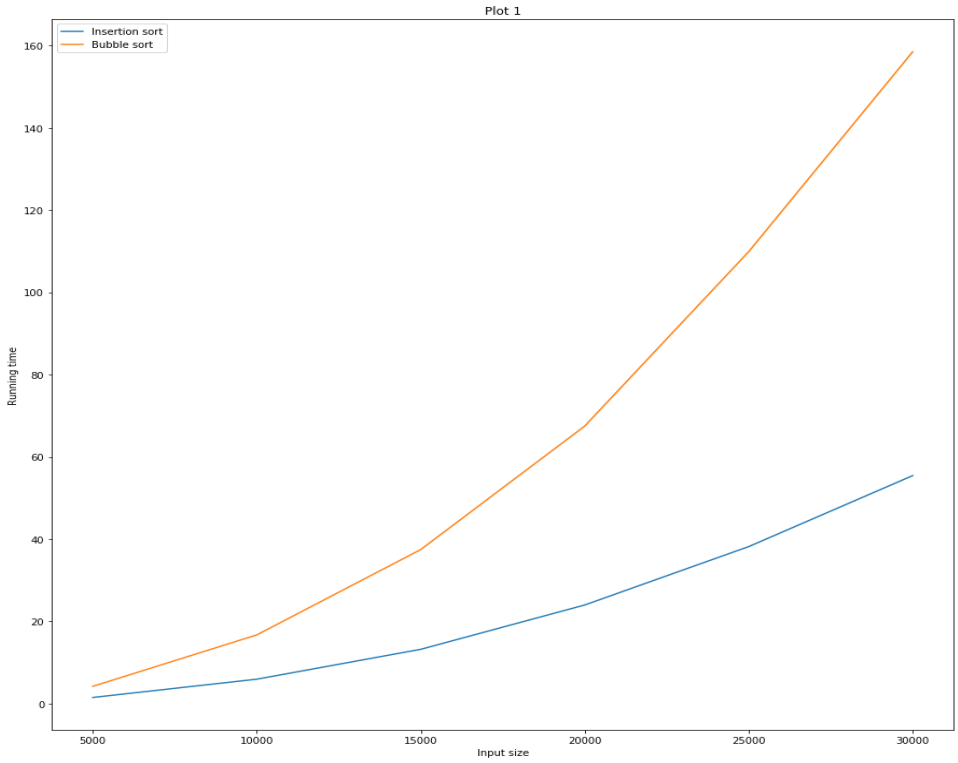
CSCI B505 SPRING 2020: Programming assignment 1

Choice of programming language- I chose python for this programming assignment as it offers excellent readability, uncluttered syntax, and has extensive support libraries- I have used one such library called matplotlib to generate the plots in this assignment.

Libraries Used- As mentioned above I have used the pyplot module of matplotlib library to generate the plots in part 1,2,3,4 of this assignment. Additionally, I have used the "random" standard library of python to generate inputs, and the "time" standard library to calculate the runtime of sorting algorithms.

Results and Interpretations:

PLOT 1 (LARGE RANDOM INPUTS):



Plot: x-axis: Input Size, **y-axis:** Run-time in seconds.

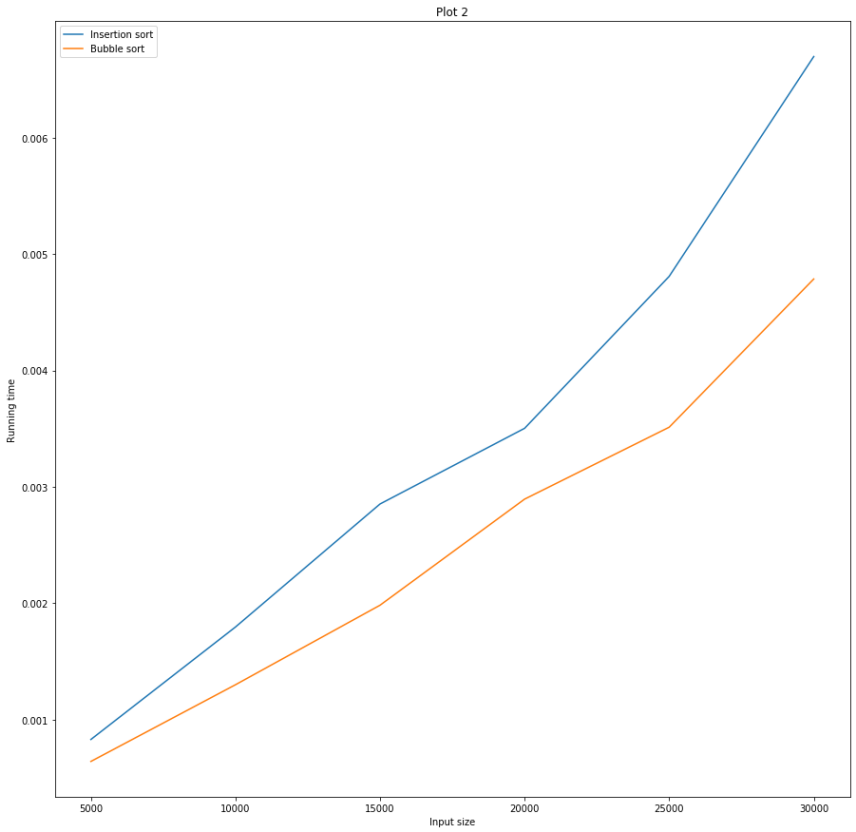
Average run-times:

1. **Insertion sort:** [1.5263986587524414, 5.963788827260335, 13.228612502415976, 24.000621159871418, 38.196101903915405, 55.44836370150248].
2. **Optimized Bubble sort:** [4.229523738225301, 16.73129177093506, 37.48156889279684, 67.48932846387227, 109.88564197222392, 158.4760014216105].

Procedure: The plot above was generated by calculating the average run time of insertion and optimized bubble sort algorithms for three input lists of equal size, where size varied from (5000,10000,15000…30,000), and the elements in each list are random integers between (1, sizeOf(list)).

Inference from plot: From the plot we can clearly observe that insertion sort outperforms optimized bubble sort algorithm regardless of the size of the input. This was expected as the optimized bubble sort algorithm runs for slightly more iterations than insertion sort but much less than normal bubble sort to check whether the list is sorted.

PLOT 2 (Non-decreasing inputs):



Plot: x-axis: Input Size, **y-axis:** Run-time in seconds.

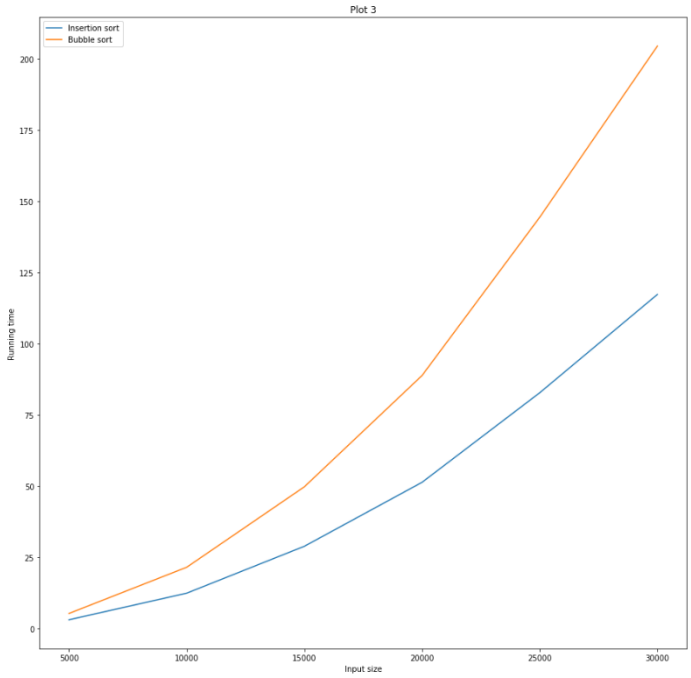
Average run-times:

1. **Insertion sort:** [0.0008299350738525391, 0.0017960866292317708, 0.0028530756632486978, 0.0035031636555989585, 0.004808584849039714, 0.00669709841410319].
2. **Optimized Bubble sort:** [0.0006412665049235026,0.0013001759847005208, 0.0019834041595458984,0.002894798914591471,0.0035125414530436196, 0.004786014556884766]

Procedure: The plot above was generated by calculating the average run time of insertion and optimized bubble sort algorithms for three input lists of equal size, where size varied from (5000,10000,15000…30,000), and the elements in each list are random integers between (1, sizeOf(list)), and sorted in non-decreasing order.

Inference from plot: From the plot we can observe that optimized bubble sort slightly outperforms insertion sort. This could have possibly been the best-case scenario for any sorting algorithm. Expectedly, both the algorithms consume linear run-time to generate the output.

PLOT 3 (Non-increasing inputs):



Plot: x-axis: Input Size, **y-axis:** Run-time in seconds.

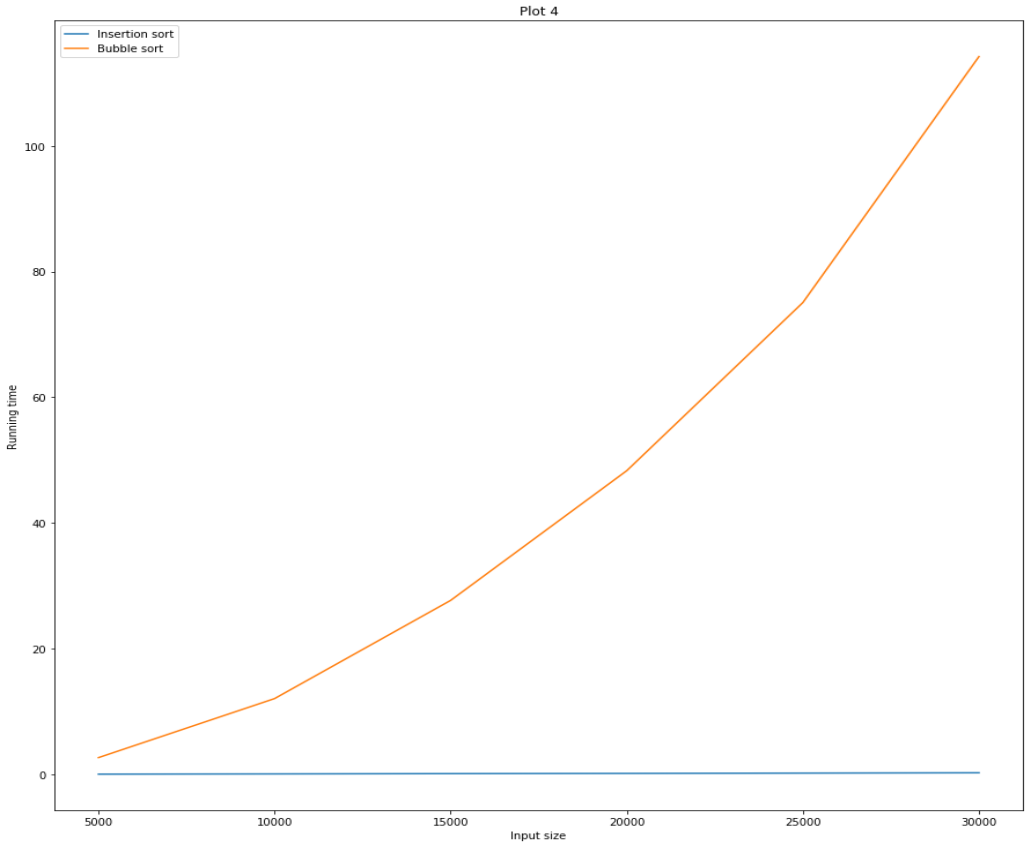
Average run-times:

1. **Insertion sort:** [3.0347280502319336, 12.361151933670044, 28.83252986272176, 51.2844603061676, 82.72192486127217, 117.24057284990947].
2. **Optimized Bubble sort:** [5.249528884887695, 21.463276465733845, 49.71267127990723, 88.77267551422119, 144.33422422409058, 204.4347280661265].

Procedure: The plot above was generated by calculating the average run time of insertion and optimized bubble sort algorithms for three input lists of equal size, where size varied from (5000,10000,15000…30,000), and the elements in each list are random integers between (1, sizeOf(list)), and sorted in non-increasing order.

Inference from plot: From the plot we can clearly observe that insertion sort outperforms optimized bubble sort algorithm regardless of the size of the input. This could have possibly been the best-case scenario for any sorting algorithm, and both the algorithms have a quadratic O(n^2) run-time complexity. This was expected as the optimized bubble sort algorithm runs for slightly more iterations than insertion sort.

PLOT 4 (Noisy non-decreasing inputs):



Plot: x-axis: Input Size, **y-axis:** Run-time in seconds.

Average run-times:

3. **Insertion sort:** [0.038671652475992836, 0.07445542017618816, 0.11959036191304524, 0.157089630762736, 0.2036773363749186, 0.2544861634572347]
4. **Optimized Bubble sort:** [2.667416572570801, 12.060670614242554, 27.686216831207275, 48.31995701789856, 75.06859882672627, 114.21112632751465]

Procedure: The plot above was generated by calculating the average run time of insertion and optimized bubble sort algorithms for three input lists of equal size, where size varied from (5000,10000,15000…30,000), and the elements in each list are random integers between (1, sizeOf(list)), and sorted in non-decreasing order with some noise- some integers are misplaced.

Inference from plot: From the plot and results we can clearly observe that insertion sort outperforms optimized bubble sort algorithm regardless of the size of the input. While the optimized bubble sort algorithm appears to follow a quadratic run-time complexity for such inputs, the insertion sort seems to follow linear time complexity- this is due to the substantial difference in run-time of both the algorithms.

INPUT 5 (Small random inputs):

Results:

- Average time of insertion sort 0.00012369340658187868, Overall run-time = average*100000= 12.36 secs.
- Average time of bubble sort with check 0.0002994989085197449, Overall run-time = average*100000= 29.9 secs.

Procedure: The results above were generated by calculating the average run-time of insertion and optimized bubble sort algorithms for a total of 100,000 arrays/lists of size 50, where the elements in lists are random integers between 1 and 50.

Inference from results: From the results we can observe that the insertion sort algorithm performs slightly better than bubble sort. I suspect its again because optimized bubble sort algorithm runs for slightly more iterations than insertion sort, though much less than normal bubble sort, to check whether the list is sorted.

Conclusion:

Both the algorithms have asymptotic quadratic running time (O(n2)).

1) **Does the first plot reflect this? How do the two algorithms compare in terms of the running time?**
   - Yes, the first plot reflects the quadratic run-time complexity of both the algorithms. From the plot we can clearly observe that insertion sort outperforms optimized bubble sort algorithm regardless of the size of the input. This was expected as the optimized bubble sort algorithm runs for slightly more iterations than insertion sort but much less than normal bubble sort to check whether the list is sorted.

2) **How about the second plot? Do you think this one is quadratic? Why do you think it looks the way?**
   - No, the second plot does not reflect the quadratic time complexity of both the algorithms. This could possibly be the best-case scenario for these algorithms. Since the inputs were already in non-decreasing order, both the algorithms have linear run-time O(n). However, the plot doesn't perfectly capture the linearity, I suspect it's due the time complexity of O(n) comparisons in both the algorithms.

3) **How does the third plot compare to the first and second?**

- Yes, the third plot perfectly reflects the quadratic time complexity of the algorithms. This could possibly be the worst-case scenario for both these algorithms. Since the inputs were in non-increasing order, both the algorithms have quadratic run-time O(n^2). From the plot we can observe that insertion sort outperforms optimized bubble sort algorithm regardless of the size of the input. This was expected as the optimized bubble sort algorithm runs for slightly more iterations than insertion sort.
- The running time for input of each size (5000,10,000,15000….30,000) is much higher in the third plot than in plot one and plot 2. The shape is similar to plot one, but dissimilar to plot two.

4) **What about the fourth plot? Can you explain the behavior?**
   - From the plot and results we can clearly observe that insertion sort outperforms optimized bubble sort algorithm regardless of the size of the input. While the optimized bubble sort algorithm appears to follow a quadratic run-time complexity for such inputs, the insertion sort seems to follow linear time complexity- this is due to the substantial difference in run-time of both the algorithms.
   - An input with only few elements in sorted form would be more favorable to insertion sort than optimized bubble sort. Because, the check in optimized bubble sort is effective only when all the elements are in sorted format, but insertion sort would skip these sorted elements in linear time. Hence, insertion sort takes substantially lower amount of time than optimized bubble sort.

5) **What about the fifth input? Note that the total input size is greater than all other inputs combined; why the algorithms can handle it efficiently?**
   - The results reflect that insertion sort algorithm performs slightly better than bubble sort. I suspect its again because optimized bubble sort algorithm runs for slightly more iterations than insertion sort, though much less than normal bubble sort, to check whether the list is sorted.
   - Both the algorithms are complete, so regardless of the size of the input we can be assured of getting the expected output eventually.

6) **Summarize in which situations these algorithms can be used in practice. Which one would you prefer?**
   - Both optimized bubble sort and insertion sort are complete, so we can be assured of getting the expected output. But both the algorithms on average have quadratic time complexities, so they are not suitable for applications that involve maintaining large data. We can use these algorithms for maintaining small scale systems and when efficiency is not the primary criterion.

- Between insertion sort and bubble, I would prefer insertion sort. As observed above, insertion sort outperforms bubble sort in cases where we had random inputs, non- decreasingly sorted inputs with noise, and non-increasing inputs, even for non- decreasingly sorted inputs, the difference in performance of both the algorithms is marginal. Also, the performance of insertion sort could be improved by using binary search for swapping the elements.