LAB REPORT : COMPSCI 2XB3

LAB SECTION – L02

Teaching Assistant - *Seyed Parsa Tayefeh Morsal*

Submitted By –

GROUP NUMBER : 2

Name : Adhya Goel

Student Number : 400280182

McMaster email : goela10@mcmaster.ca

Name : Mridul Arora (Contact Member)

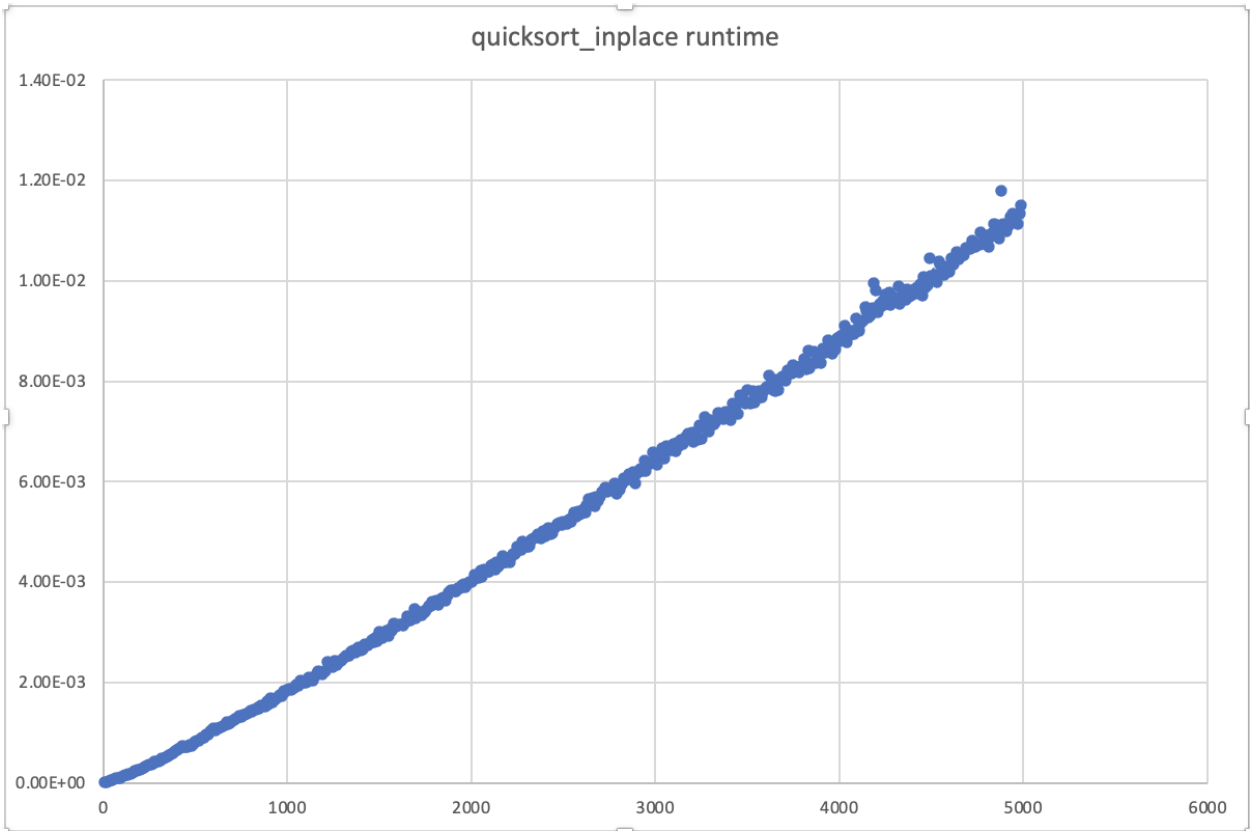
Student Number : 400253526

McMaster email : aroram15@mcmaster.ca

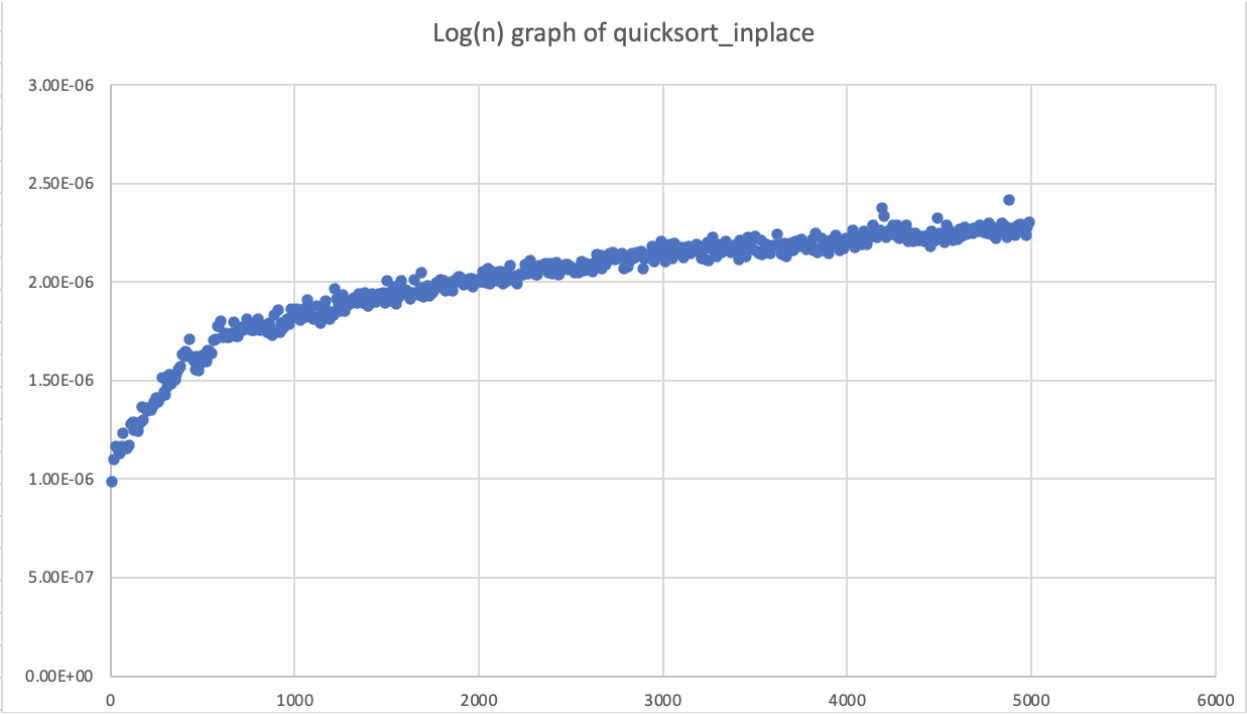
In-Place Version

In quicksort_inplace() function it has two advantages over the given implementation. First, it sorts in place, and no additional storage is required. Second, use of temporary variables rather than lists is easier to implement as no copies of lists are made in in-place version.

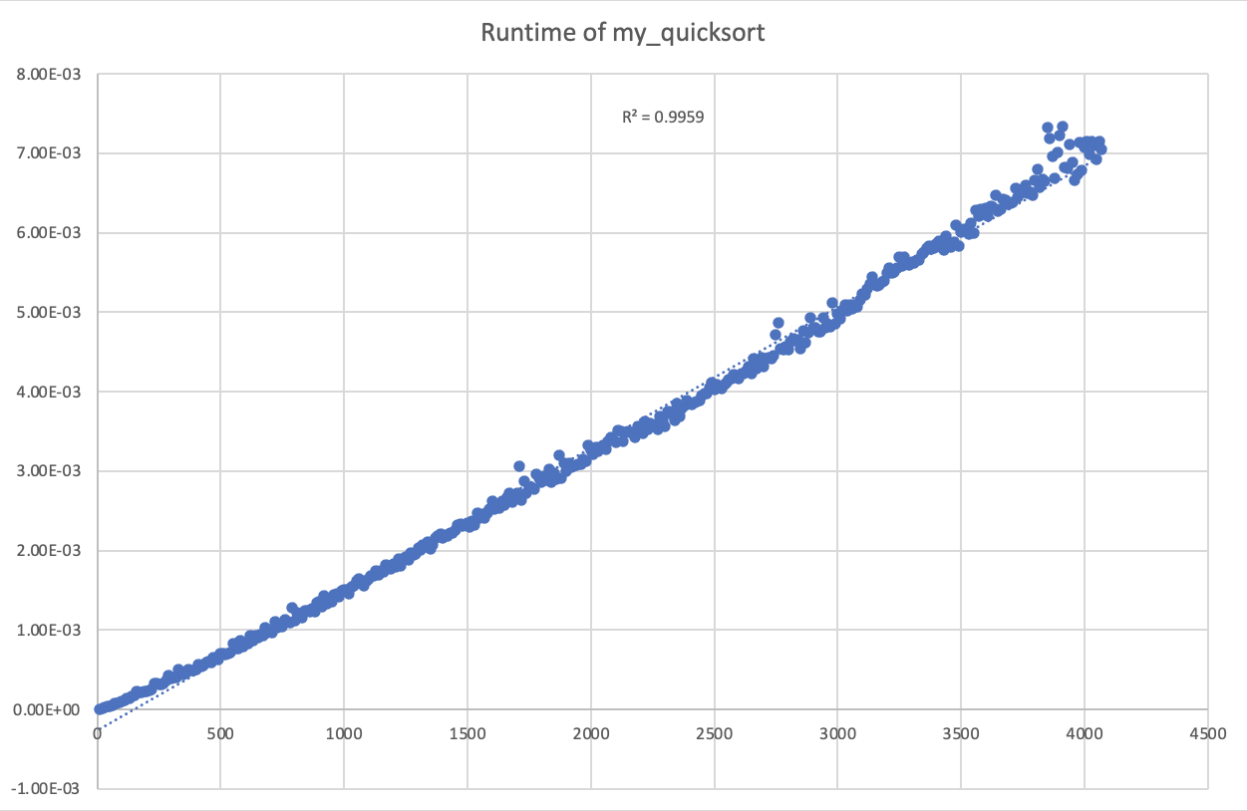
After analyzing, the runtime of quicksort_inplace() the trendline obtained looks like linear. This trendline could mislead anybody as it can be a graph of nlog(n).



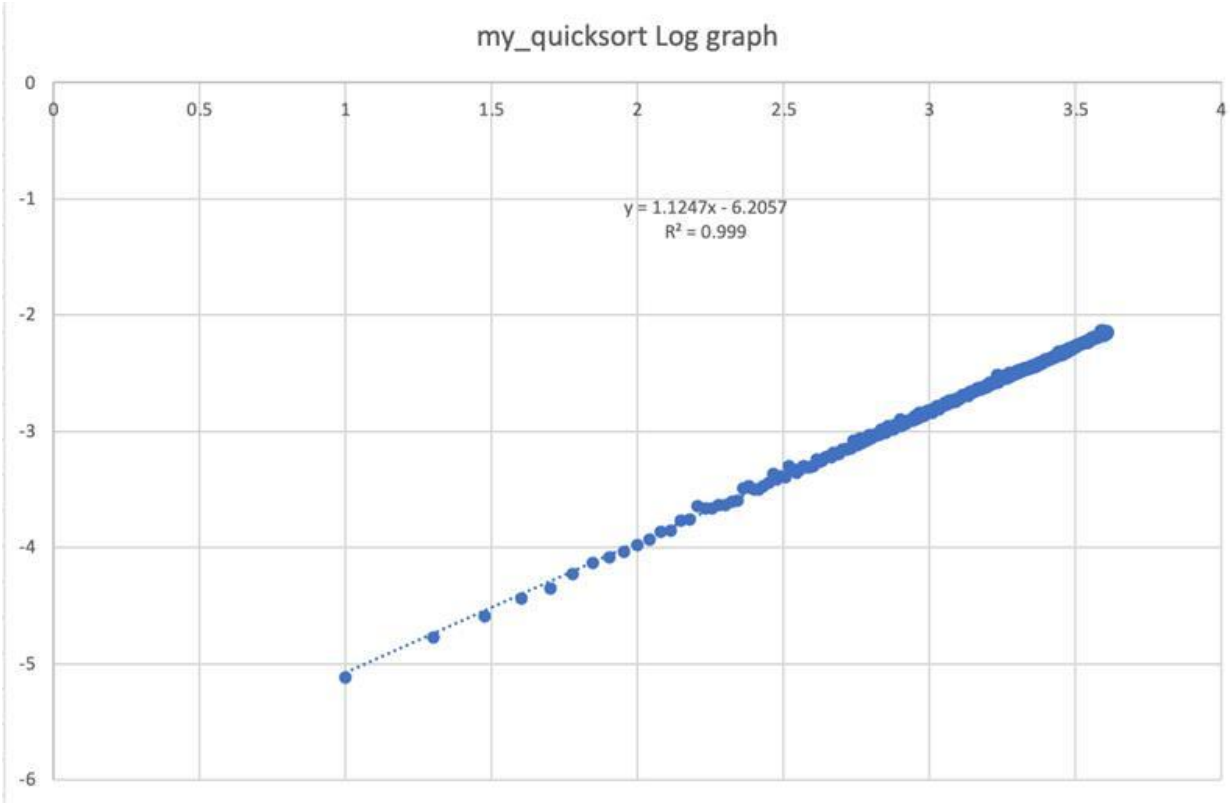
So, to obtain the actual runtime graph for quicksort_inplace() divide the runtime by n to get the graph of log(n) where n is the length of the list. And, indeed after experimenting the graph came out to be log(n).



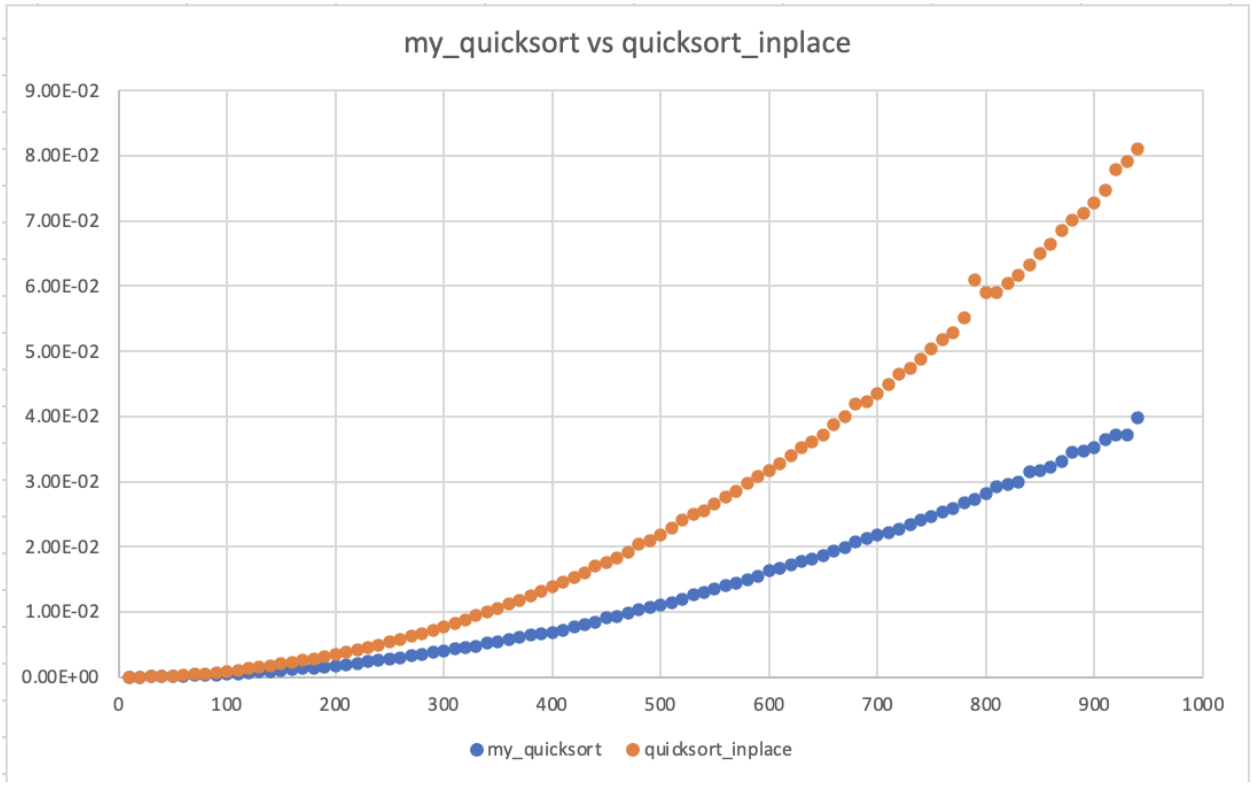
For function my_quicksort(), after analyzing the runtime data the following graph is obtained. Looking at the function this graph could be a polynomial, exponential or linear. So, to get the right result further experiments are required.



So, by calculating the slope of log(n) function of data points as determined in lab 02 the following graph was obtained for function my_quicksort(). The slope of the following graph came out to be approximately 1 which suggested that the function is a polynomial of degree 1 which is ultimately linear.



Now, after analyzing the runtime for both of the functions, my_quicksort() has better implementation than quicksort_inplace() by n(log(n) -1) where n is the length of the input list.

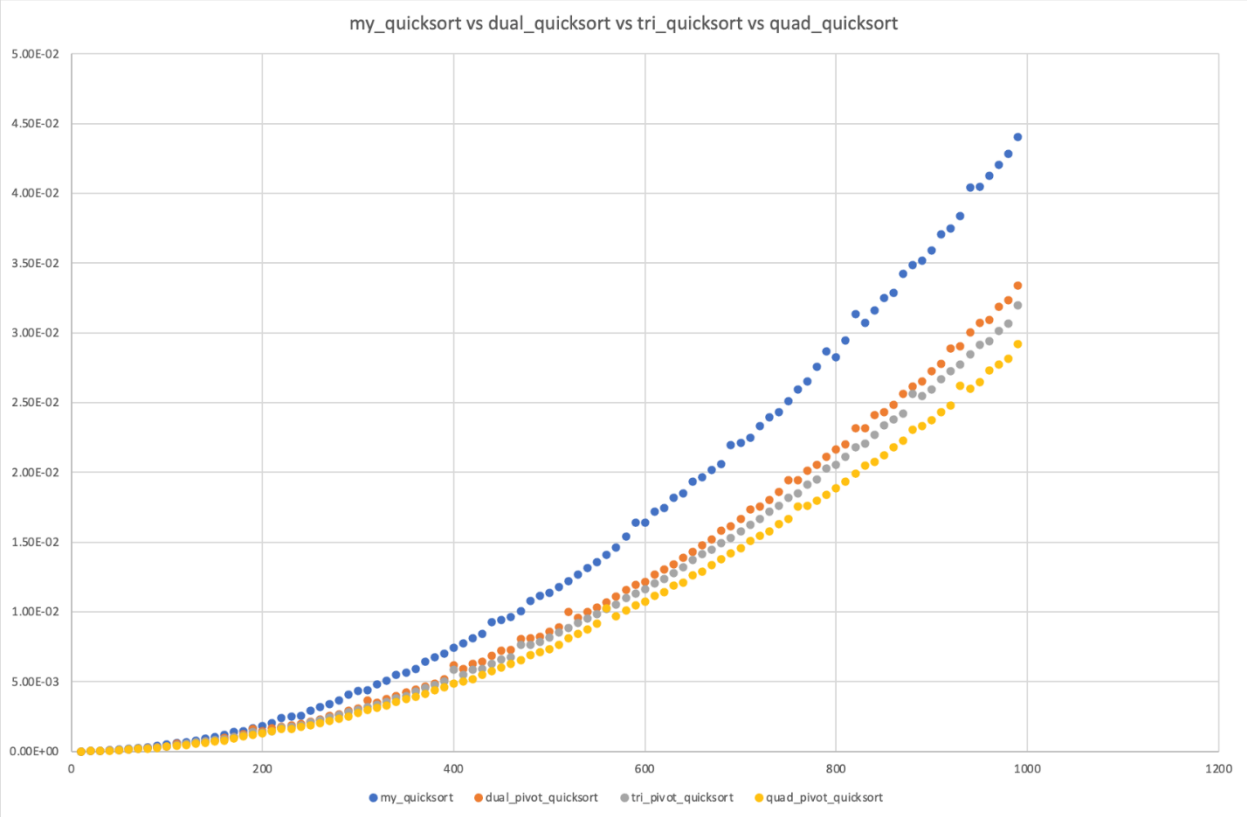


From the graph above and doing further experiments using factor function it was concluded that my_quicksort() would be preferred as it requires less time to get executed than quicksort_inplace(). While doing the experiment it was noticed that quicksort_inplace() function was taking almost double the time as compared to my_quicksort().

Multi-Pivot

Out of the four variants my_quicksort(), dual_pivot_quicksort(), tri_pivot_quicksort(), and quad_pivot_quicksort() from the graph it is recommended to use quad_pivot_quicksort().

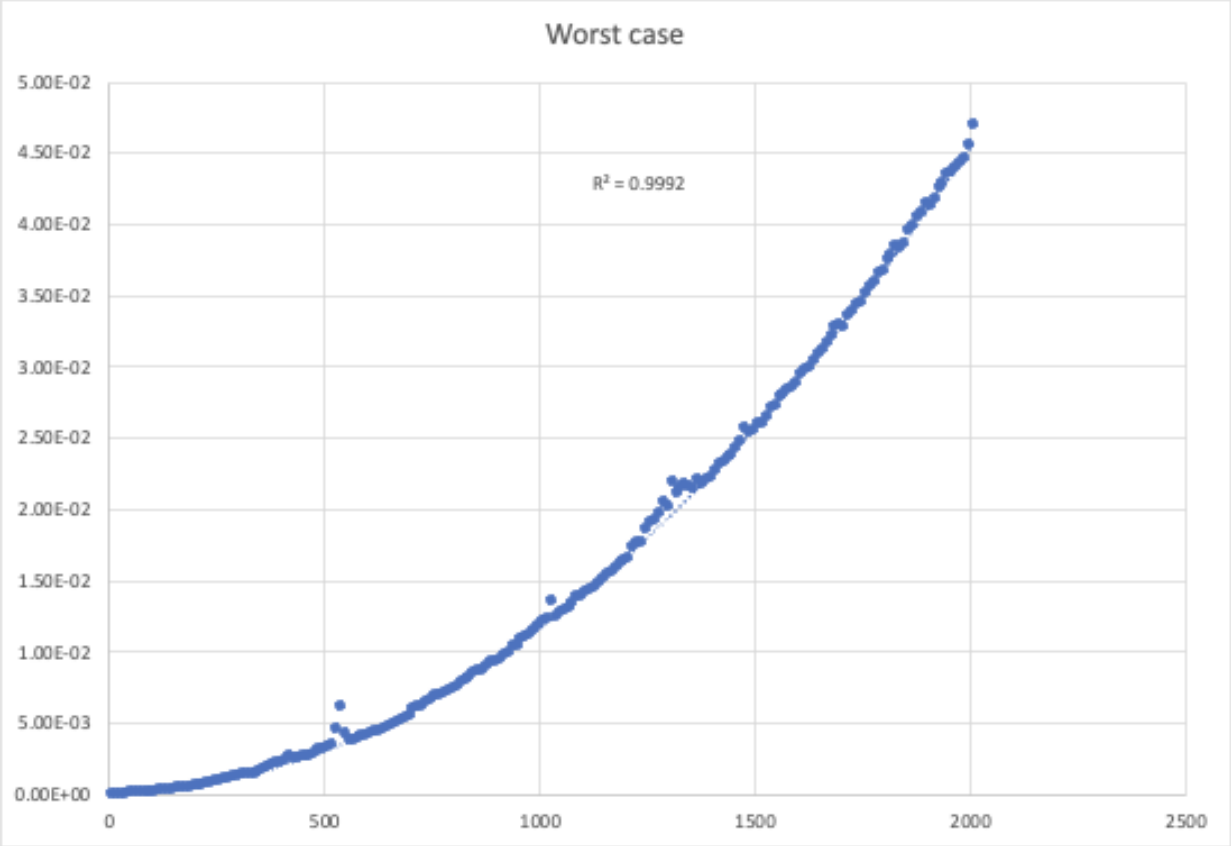
From now on, by default quicksort refers to **quad_pivot_quicksort()**.



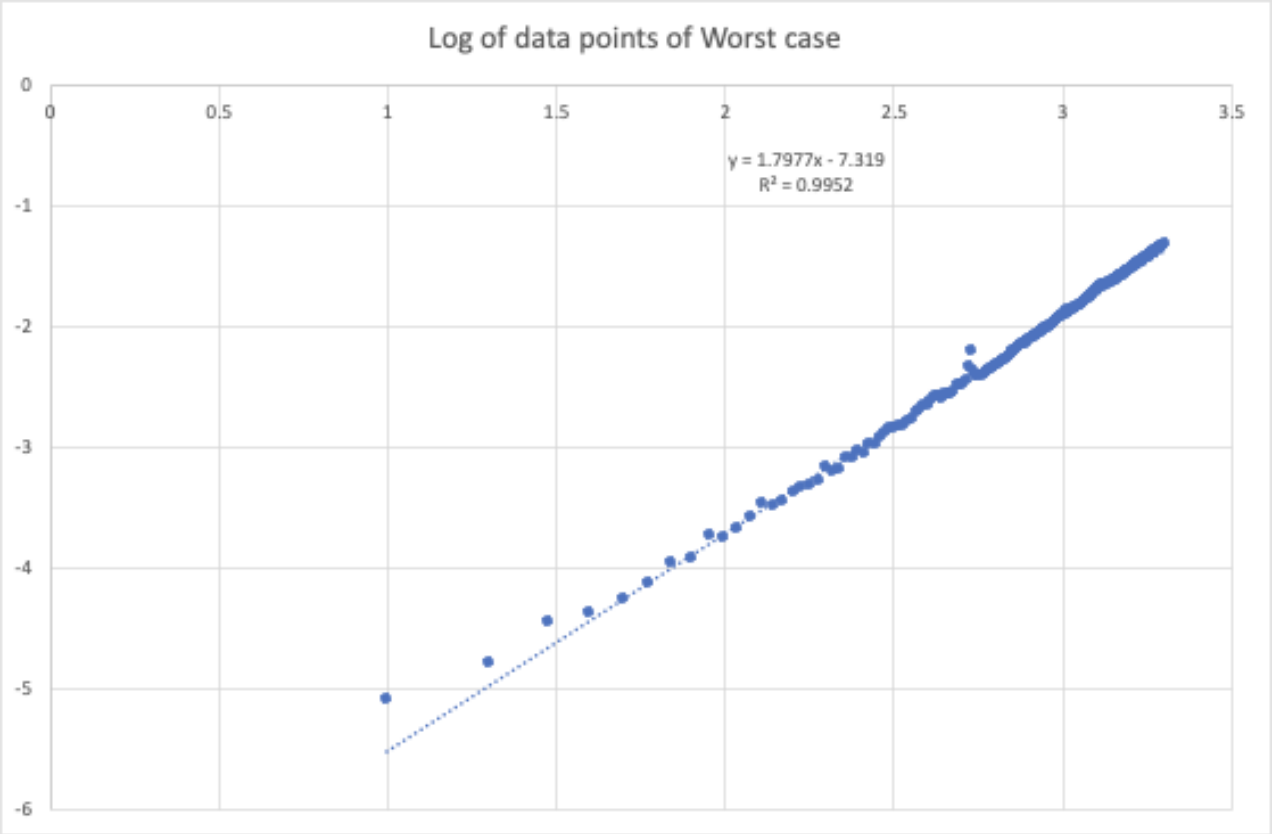
Worst case Performance

The worst case time complexity of quicksort is O(n^(2)). This happens when input array is reverse sorted. Also, it can happen if either first or last element is picked as pivot.

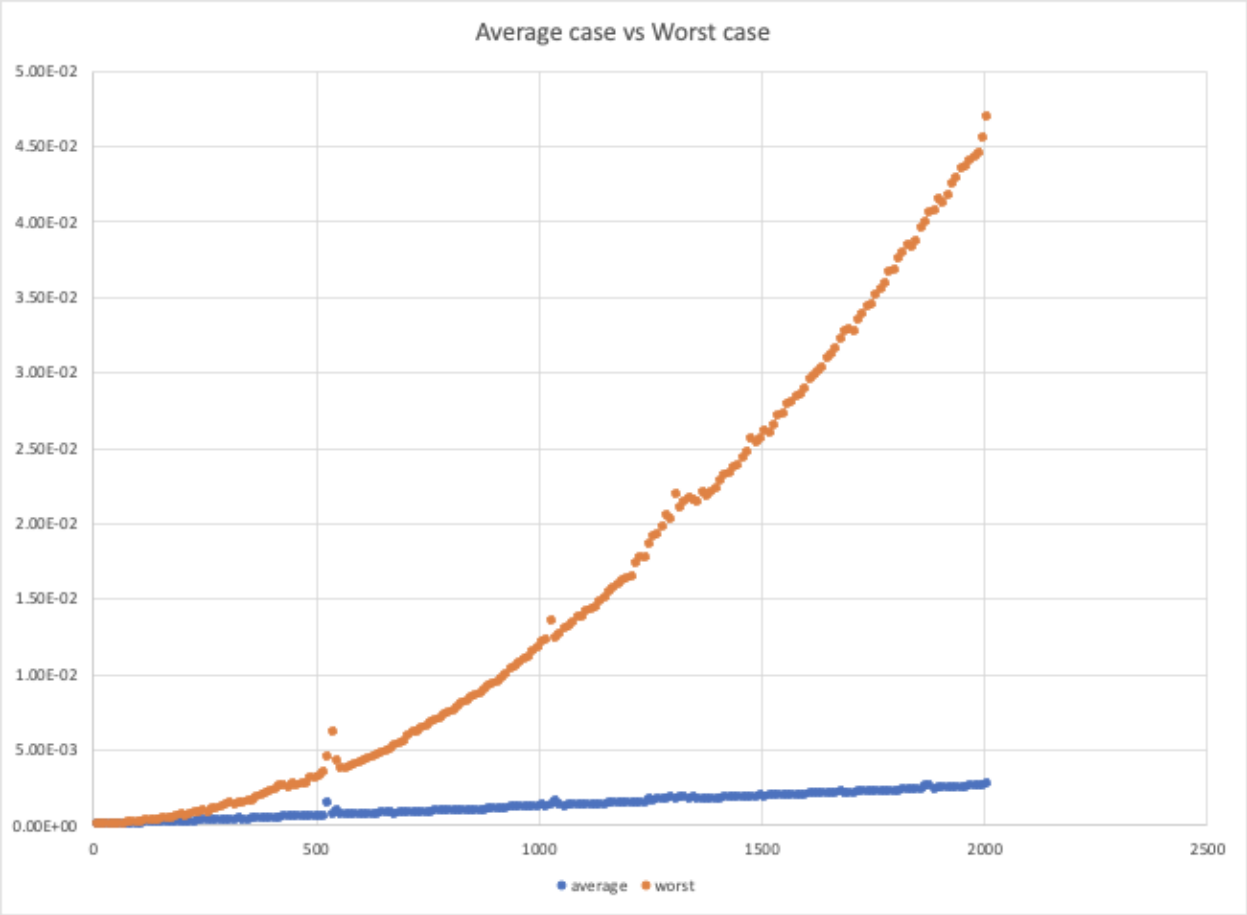
Since the time complexity of worst case looks like O(n^2)) from the following graph.



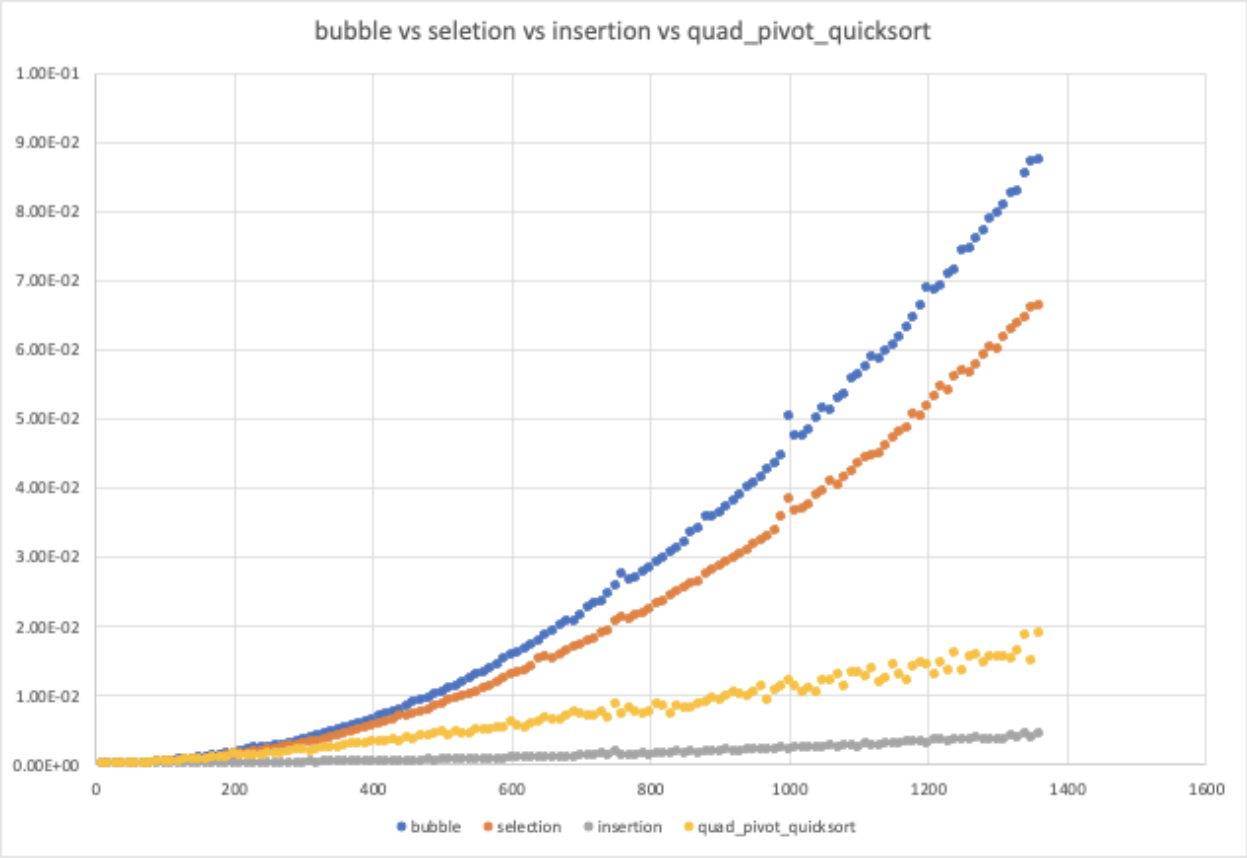
So, to confirm the previous result a graph was plotted for log of data points as learnt in previous lab. Thus, it is concluded that the graph for worst case is O(n^2)).



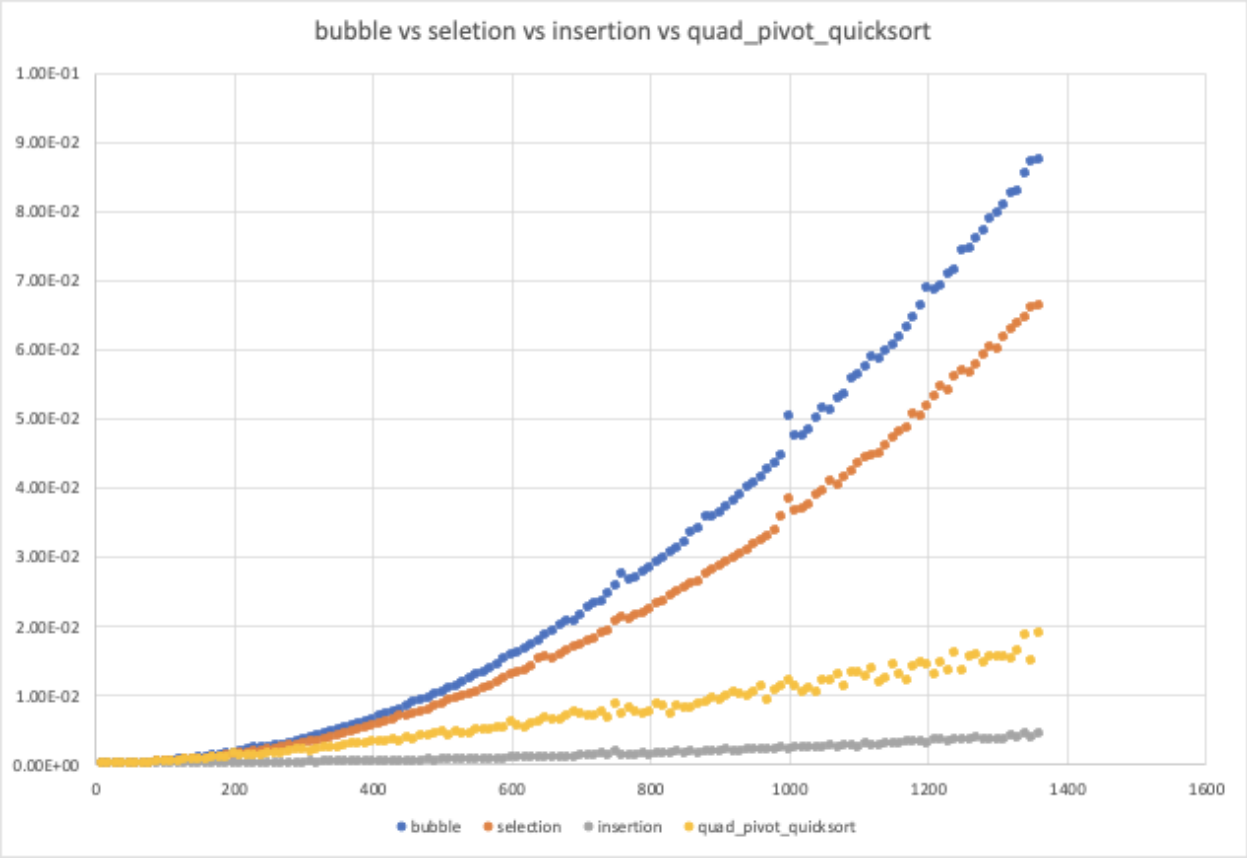
The following graph represents average case vs worst case vs n for quad_pivot_quicksort() implementation.



When experiment was run with factor 0.01 and length of the list was more than 1000 it was observed that insertion sort out of all the elementary sorts outperforms the quad_pivot_quicksort().

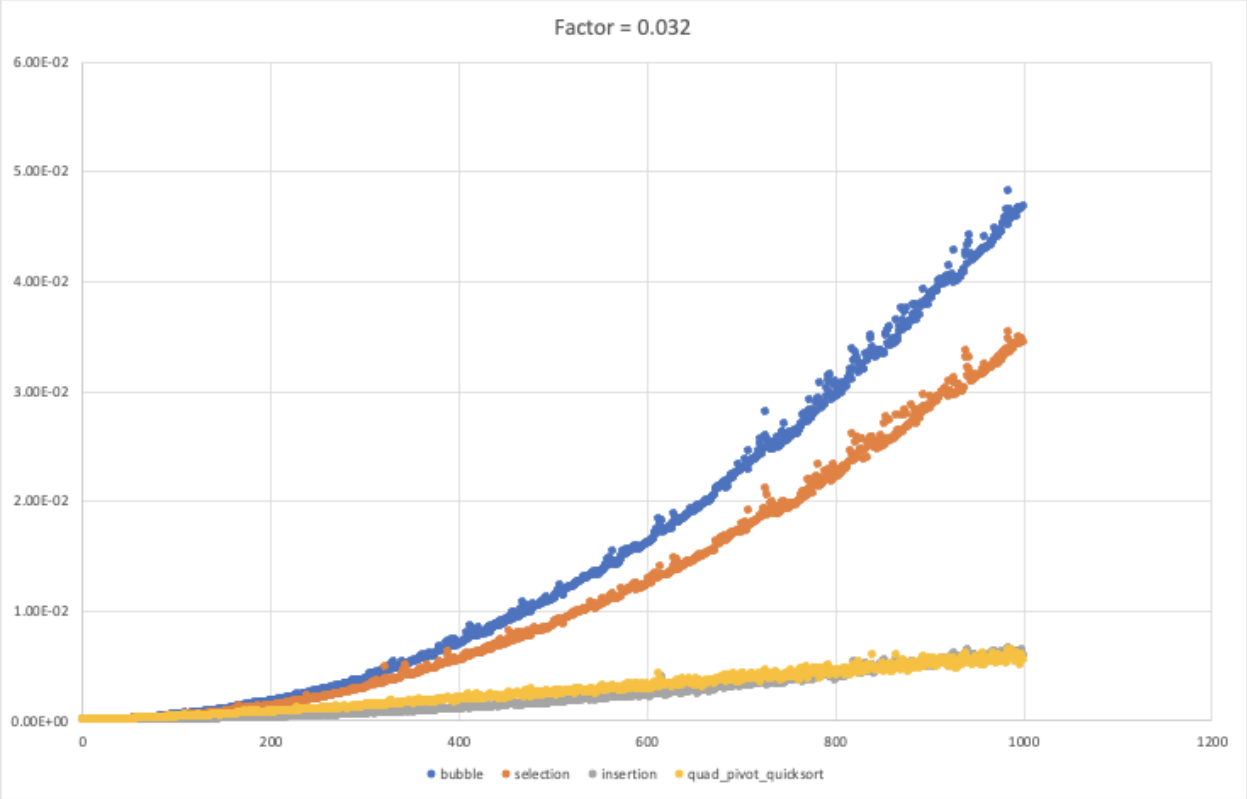


Similarly, the experiment was done on all the four sorting algorithms with factor 0.01 and length of the list 1000. It was observed that insertion sort works quicker than all the other algorithms when the list was almost sorted.

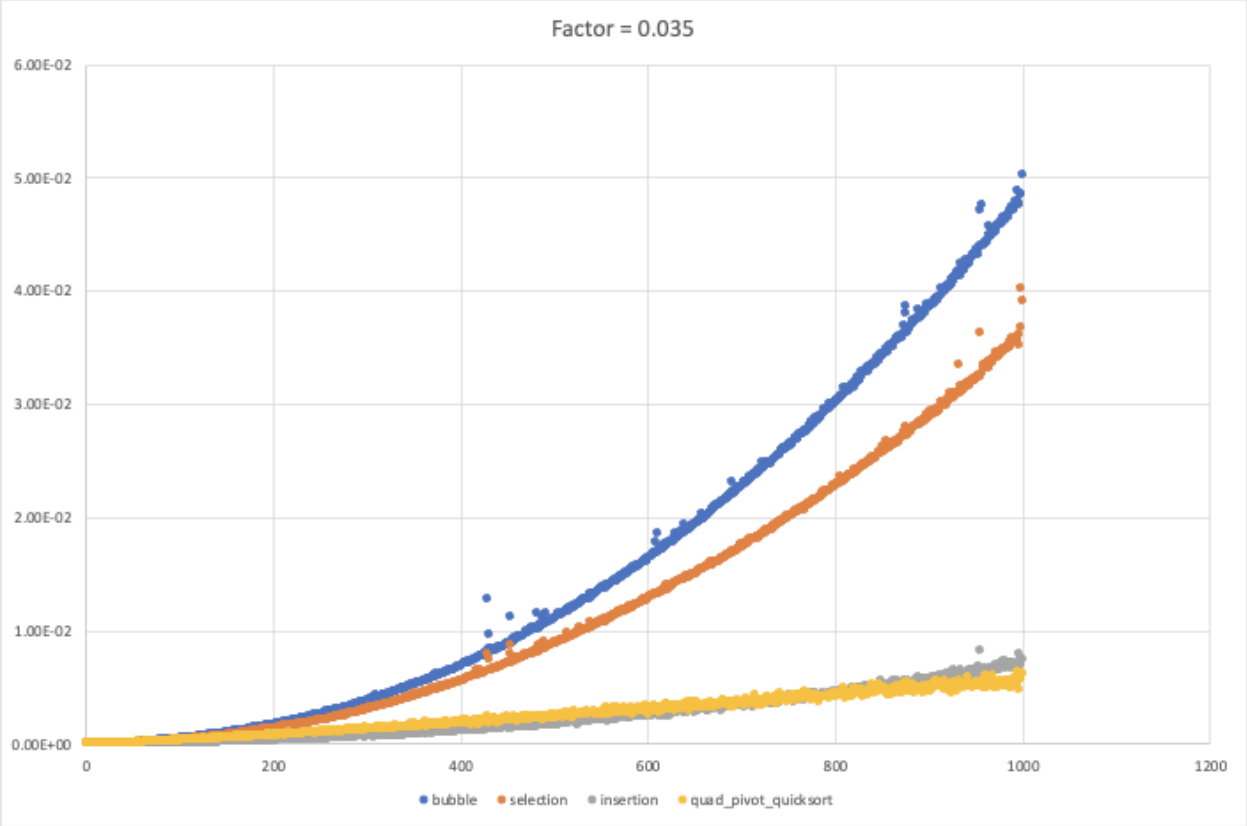


When the factor was taken to be 0.032 it was observed that the quad_pivot_quicksort and insertion sort took the same time to get executed.

To prove that quad_pivot_quicksort outperforms other algorithms at some value of factor. So, for slightly greater than 0.032 value another experiment was conducted.

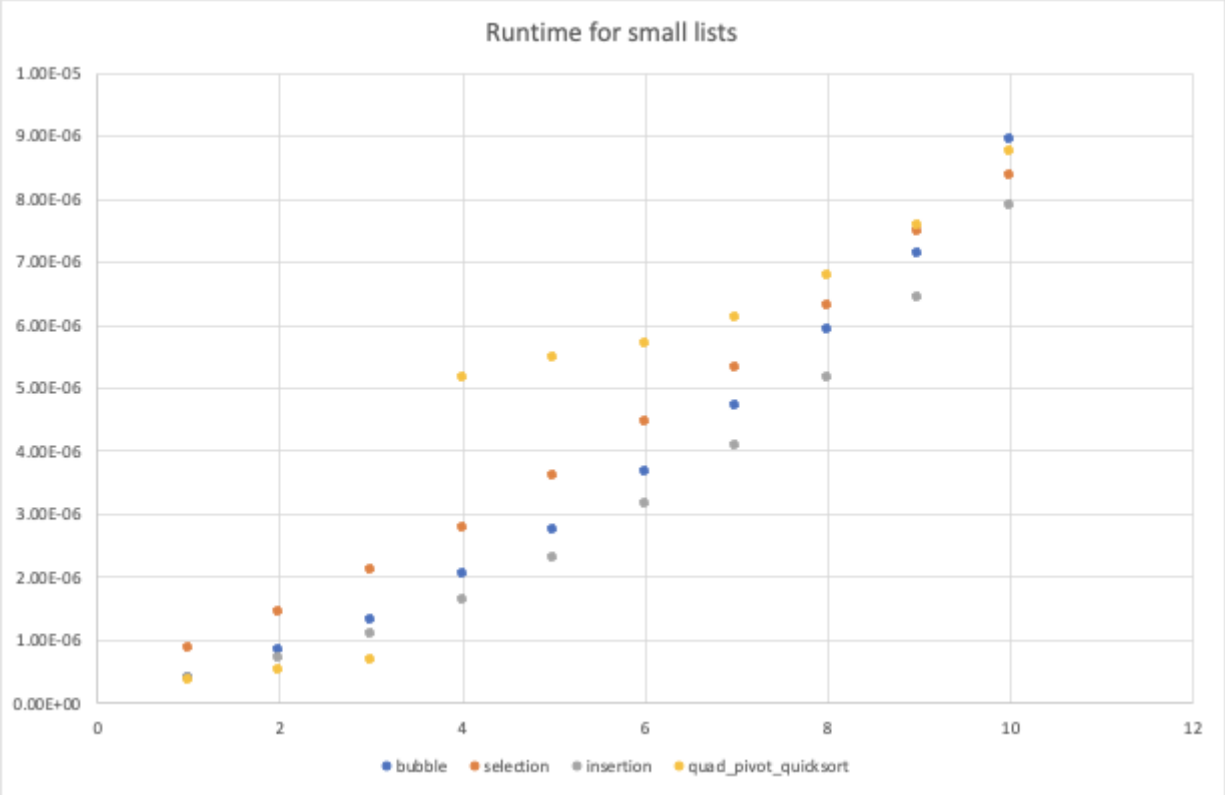


On value 0.035 for factor quad_pivot_quicksort outperformed the other algorithms.



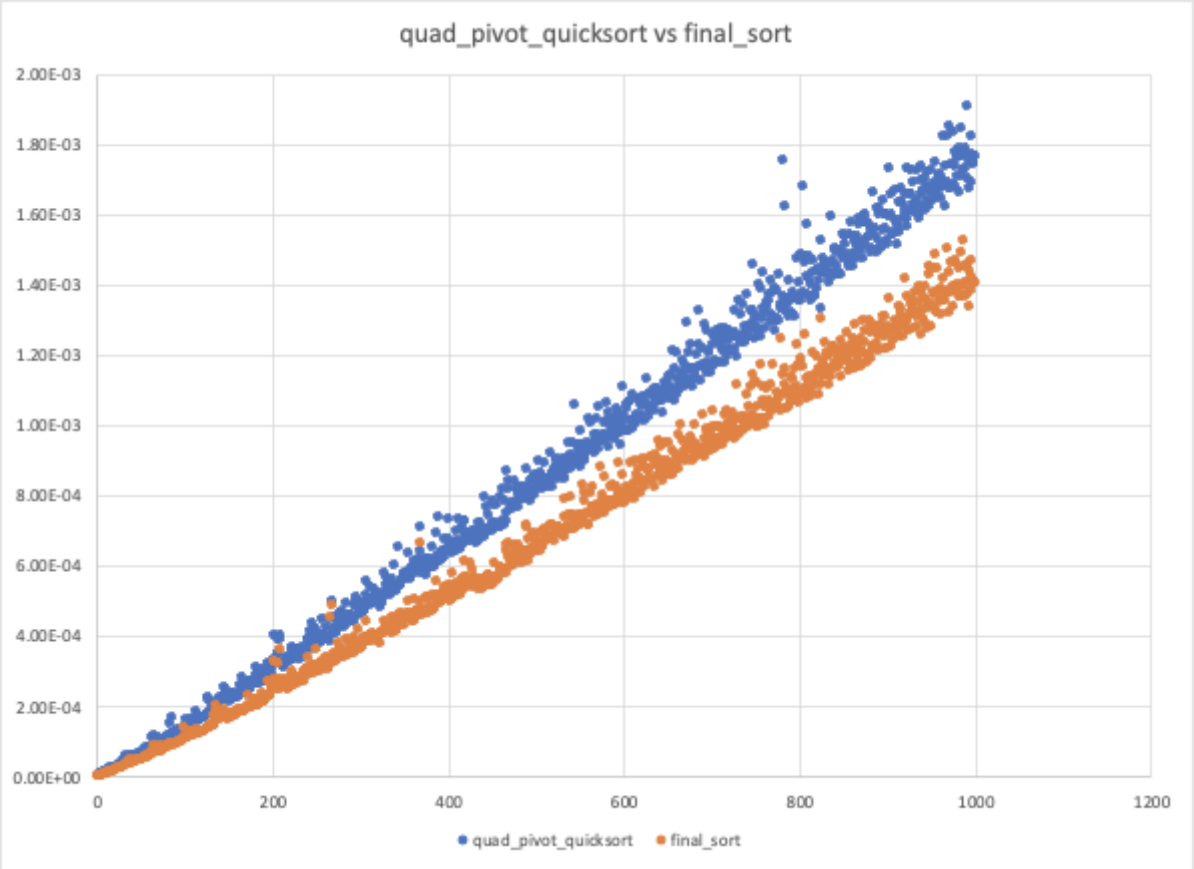
Small Lists

For small lists (where n = 10) we observed that insertion sort performs the best among all the other sorting algorithms.



When designing the final_sort hybrid function it was considered that for small lists insertion sort performs the best as showed above and from all other multi-pivot sorting algorithms quad_pivot_quicksort works the best when the size of the list is big as proved earlier. Hence, hybrid function consists of the two most accurate sorting algorithms with fastest running complexity as compared to other ones.

Final goal, expected for the final_sort which is the hybrid function will perform the best i.e will get executed in less time when compared with the default recommended algorithm (which is quad_pivot_quicksort).



REFERENCES

- *https://www.geeksforgeeks.org/python-program-for-bubble-sort/*
- *https://iopscience.iop.org/article/10.1088/1757-899X/180/1/012051/pdf*