Fangting Ma (fm128)
Kaifeng Yu (ky99)

Report for Project 1 – Sorting Algorithms

We have implemented five sorting algorithms including selection sort, insertion sort, bubble sort, merge sort and quick sort in python, got the log-log slope for those algorithms and some plots of runtime versus input size for those five sorting algorithms compared to the python built-in sort function.

Overall, these five sorting algorithms behave as expected for both unsorted and sorted inputs lists. It can be seen from the table below that, for random data, the first three sorting algorithms all have a log-log slope around two, which is consistent with their big-O time complexity of $O(n^2)$. Since selection sort has the same time complexity for all the cases, it is expectedly that the log-log slope for sorted data is also about two. While the log-log slope of insertion sort and bubble sort is nearly one for sorted data, because if the data are sorted, they will be in their best-case runtime, $O(n)$. Among these five algorithms, it is noticeable that merge sort has a log-log slope a little over one for both random and sorted data. It meets our expectations since merge sort has the same performance for all the cases, whose runtime is $O(nlogn)$. Quick sort behaves similarly to merge sort when dealing with random data, while the log-log slope for sorted data is approximately two, which means it is in the worst case for this algorithm. It is not surprising, because our quick sort chooses the first element of list as the pivot value.

Table: log-log Slope for different sorting algorithms (Averaging over 30 Trials)

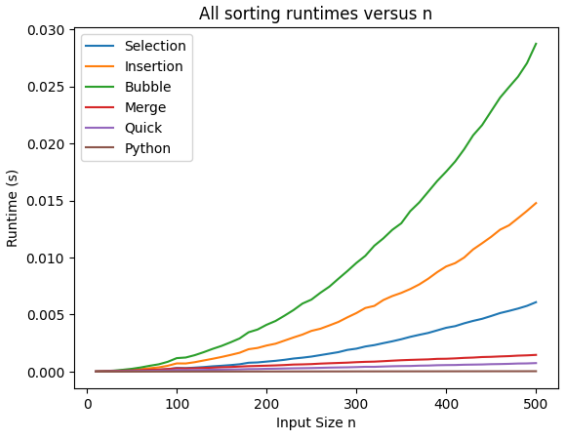
	random data		sorted data	
	all n	n>200	all n	n>400
Selection Sort	1.757971	2.163471	1.941543	2.076367
Insertion Sort	1.917691	2.044615	1.031751	1.027307
Bubble Sort	2.006759	2.140757	1.030173	1.022562
Merge Sort		1.131128		1.110071
Quick Sort		1.213134		2.053756

In our view, quick sort is the best among these five sorting algorithms for three main reasons. The first reason is that it has the time complexity of $O(nlogn)$ for both best and average cases, which is better than the first three algorithms whose average runtime is $O(n^2)$. They will behave much slower than quick sort and merge sort if they are used to sort large lists, which will be common in practice. Another reason is that it is an in-place sorting algorithm. Although it seems that merge sort has the better performance for worst case than quick sort, merge sort is an out-of-place algorithm. It means that it would take much more memory than quick sort and negatively influence the runtime for especially large lists. It can be seen from the figure 1 below that for larger data, quick sort would perform better than merge sort. The last reason is that though it has the worst case of time complexity $O(n^2)$, it would not be a common case and we can design some strategies to sensibly choose the pivot. Therefore, quick sort would be a better choice when sorting data under most situations.

Besides, it is obvious in the figure 1 that bubble sort would be the slowest sorting algorithm among these five algorithms, though it has the same big-O complexity of $O(n^2)$ as insertion sort and selection. It possibly due to the significant times of swaps to place each element into their right place, while the other two algorithms only swap once for each element.

Fangting Ma (fm128)
Kaifeng Yu (ky99)

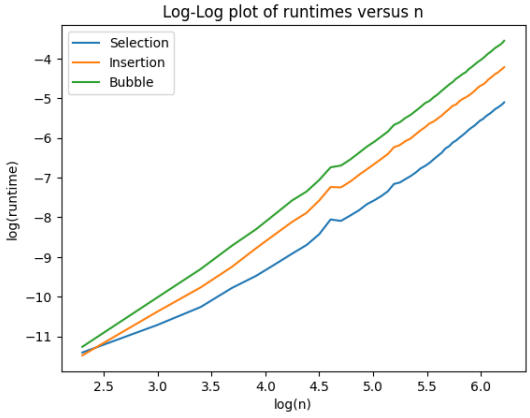
Figure 1. All sorting runtimes VS input size for random data (left)



When calculating the theoretical runtimes, we usually use relatively large values of input sizes for two main reasons. The most significant reason is that big-O complexity mainly concerns about the behaviour of the highest cost term when input size becomes large, which also means it cares more about which algorithm grows faster with the increase of input size instead of the runtime for a specific input size. For small input sizes, the influence of coefficients and all lesser terms is relatively large. We can only ignore them when input size is quite large. The other reason is that for small input size, the runtime for all the algorithms will be small and it does not make much difference. It can be seen from figure 1 that for those five algorithms, the differences of runtimes for small input size are small.

For small input sizes, the runtime would not be consistent with big-O complexity, since smaller terms and coefficients might have a larger impact on runtime than the highest cost term. As can be seen from the figure 2 below, for smaller input sizes, selection sort has a larger runtime than insertion sort, although it has smaller runtime for large input sizes.

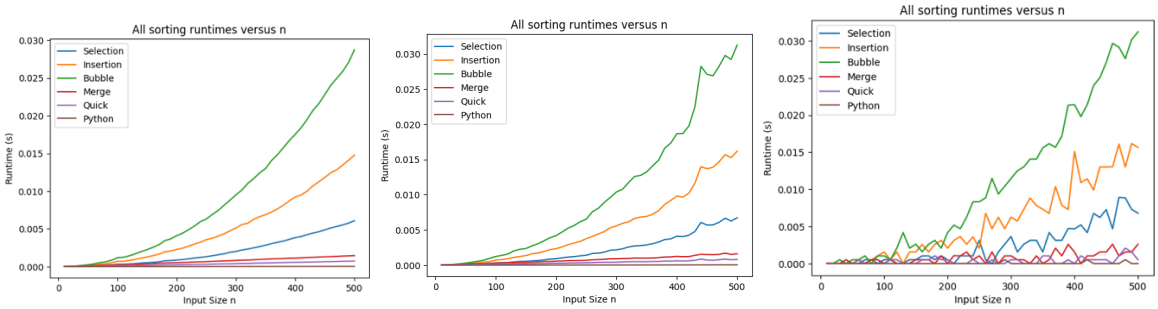
Figure 2. Log-Log plot of runtimes VS input size for random data



In order to minimize the influence of other factors unrelated to asymptotical complexity, it is reasonable to average the runtime across multiple trials when we try to get experimental runtimes. For example, the runtime could be sometimes slower than expected if the computer is running other computationally expensive task in the background at the same time, as shown in the figure 3 below. The plots in the middle figure are a little unstable than the left figure. It

may be due to the less CPU allocation when running multiple tasks, which will slower the programs. We also operated our program on different operating systems. According to the plots in the figure 3 below, it is noticeable that the plots fluctuate a lot, and the log-log slope is the largest for running on a Windows operating system. Therefore, it is necessary for us to average runtimes over multiple trials instead of only relying on one specific trial, which might lead to inaccurate results.

Figure 3. All sorting runtimes VS input size for random data on MAC (left),
All sorting runtimes VS input size for random data on MAC while the computer performs other tasks (middle),
All sorting runtimes VS input size for random data on a Windows computer (right)



Since there are many noise factors influencing experimental runtimes, we usually analyse theoretical runtime when we only want to compare the sorting algorithms with respect to their algorithmic performance. Besides, it is not realistic to get the experimental runtime for extremely large input size since it would take long time. In this case, analysing experimental runtimes would also be helpful. However, the experimental results are also valuable under some circumstances. When two algorithms have the same big-O complexity, we might want to see if the practical runtimes of them are also the same, then doing the experiments would be necessary. Getting experimental runtimes will reveal some other operational factors which may also influence the runtime, such as memory allocation. For example, merge sort and quick sort have the same runtime of $O(nlogn)$, while the experimental results reveal that merge sort will take more time for the same large input size than quick sort.