Project1

Zihao Lin (NetID: zl293)

Oct. 8th 2020

In this project, I compared five sorting algorithms, selection sorting, insertion sorting, bubble sorting, quick sorting and merge sorting, by python. According to theoretical analysis, the theoretical run time of each sorting algorithm is shown below:

Algorithm	Insertion	Bubble	Selection	Quick	Merge
Best case	$O(n)$	$O(n)$	$O(n^2)$	$O(n\log n)$	$O(n\log n)$
Average case	$O(n^2)$	$O(n^2)$	$O(n^2)$	$O(n\log n)$	$O(n\log n)$
Worst case	$O(n^2)$	$O(n^2)$	$O(n^2)$	$O(n^2)$	$O(n\log n)$

We can see that in the average case, merge sort and quick sort may be better than the other three. However, in the best case, bubble sort and insertion sort are the best, quick sort and merge sort are slower than them, and the worst algorithm is selection sort.

The following plot shows one table in which are the run time of each sorting algorithm in the biggest number of n (smaller than 1000). the runtime of the five algorithms and one python sort. Figure 1 is the run time of sorting unsorted lists, and figure 2 is the run time of sorting sorted lists. According to the plots and the runtime table, we can conclude that in the average case, quick sorting and merge sorting are faster than bubble sorting, selection sorting and insertion sorting, which is same as what we expected according to theoretical analysis. In the best case, the result also satisfies what we expected. In the plot, the run time of selection sort is the slowest one and quick sort is the second slowest algorithm. Others are close and all very fast, however, the merge sort algorithm is a little slower than insertion and bubble sort. *(Answer question 6.3-1)*

Algorithm	Insertion	Bubble	Selection	Quick	Merge
Best case	0.000417	0.000133	0.050337	0.042039	0.003956
Average case	0.014830	0.043234	0.014161	0.001162	0.002694

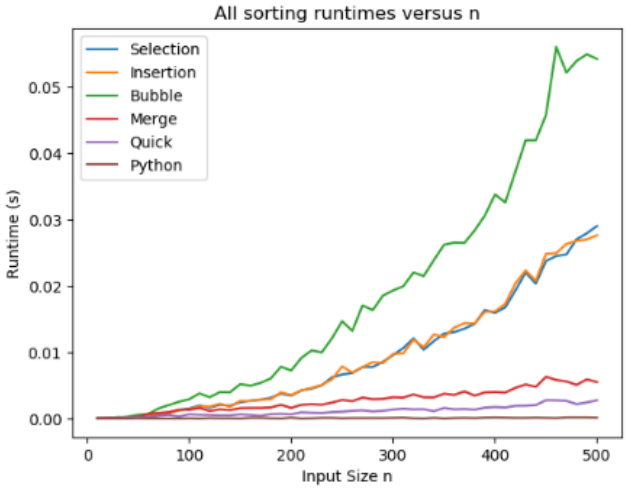

Figure 1

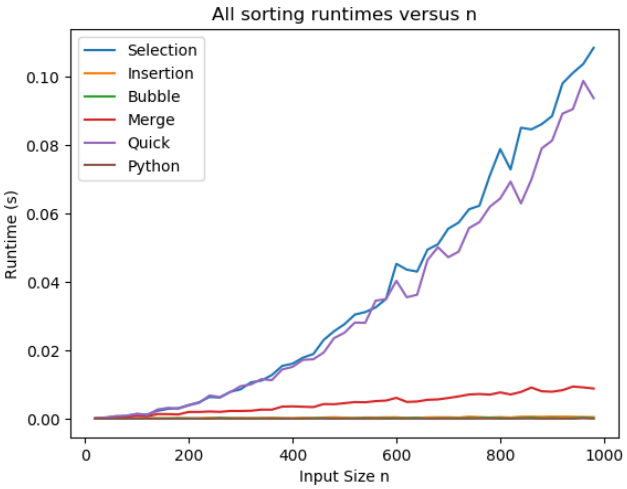

Figure 2

We look at some details in these plots and the table. In unsorted case (figure 1), the best algorithm except python sort is quick sort whose run time with biggest n is 0.001162 and the worst algorithm is bubble sort whose run time with biggest n is 0.043234. In the sorted case, the best algorithm is bubble sort which run time with biggest n is 0.000133. Insertion sort is the second-best algorithm whose runtime is not so different from bubble sort. The worst algorithm is selection sort whose run time with biggest n is 0.050337. That's reasonable. First, in unsorted case (figure 1), bubble sort has the most swap times and has to search many values, however quick sort doesn't need so many swap times. In the sorted case (figure 2), run time depends on the search time. The best algorithms do not need to swap, and their search time is $O(n)$ while others are larger than that. The worst case, quick search needs $O(n^2)$ to search numbers. *(Answer question 6.3-2)*

However, when we compare different algorithms, we assume that the number of n is very large. Only when n is large enough, the difference between algorithms is obvious, which in mathematical means that the difference among n^2, $n \log n$ and n is obvious. We can see that in the plots above, when n is small, there is not obvious difference between algorithms. That's because each algorithm

does not need too many searching actions or swap actions. The difference among n^2, $n\log n$ and n is small. *(Answer question 6.3-3 & 6.3-4)*

Another thing I want to mention is that we use the average time across multiple trails to compare each other, which is because this will reduce the influence of some particular case. For example, due to the different logic of algorithms, there may be one list that need the same time to be sorted by selection sort and bubble sort. If we just use this one trail, our conclusion will be that the run time of bubble sort and selection sort are same. However, that is not true. *(Answer question 6.3-5)*

Running code will consume a lot of memory. If I want to open a browser when I am running some complicated code, my computer will be very slow because there is not enough memory for the computer to open a browser. *(Answer question 6.3-6)*

Theoretical analysis gives us a result which enables us to better understand the essence of the algorithm excluding all external factors and plays a very important role in our design of a program. However, experimental runtimes are also very important because in the real industry, some applications or some systems' speed depends on the experimental runtimes but not the theoretical time. *(Answer question 6.3-7)*

Last thing is log-log runtime plot. We can see that the larger the n, the better the line fits. We have a function which is $\log T = k\log n + \log a$. When n is very small, the real run time will not change obviously (see in figure 1 & figure 2, n is lower than 200). In this situation, the runtime depends more on the computer performance but not n. Therefore, $\log T$ (here T is real runtime and $k\log n + \log a$ will not match which will influence the result of fitting. However, when n is very large, the runtime will much depend on n. In this situation, the runtime will depends more on n but not computer performance. So, I think fitting log-log runtime using only larger values of n is better than using all values of n. *(Answer question 6.2)*