Project 1: Sorting

Zedong Gao (zg79), Yunbo Liu (yl815)
27th October 2021

This report addresses several questions based on the results of our sorting program.

Question 1

Selection Sort: As shown in Figure 1 and Figure 2, both the slope of the line representing Selection Sort is 2, which means that its runtime is O (n^2) for both unsorted and sorted input arrays. It is exactly what we expected because Selection Sort performs the same runtime - O (n^2), regardless of the input is a best case or a worst case.

Insertion Sort: We can see that the slope of the line representing Insertion Sort is 2 in Figure 1 while 1 in Figure 2, which indicates that the runtime of Insertion Sort is O (n^2) for unsorted input arrays and O (n) for sorted input arrays. It makes sense because for randomly generated input arrays, as average cases or worst cases, each insertion costs O (n) work, with n total insertions for a total of O (n^2) runtime. While for already sorted input arrays, which are best cases, each insertion only costs O (1) work, with n total insertions for a total of O (n) runtime.

Bubble Sort: Similarly, In Figure 1, the slope of the line representing Bubble Sort is 2. While in Figure 2, the slope of the line representing Bubble Sort is 1. It means that the runtime of Bubble Sort is O (n^2) for unsorted input arrays and O (n) for sorted input arrays. It meets our expectation because for unsorted input arrays, as average cases or worst cases, each iteration takes O (n) work, with total O (n) iterations for a total work of O (n^2) runtime. However, for those already sorted input arrays, which represents best cases, no swaps are made and only takes one pass of O (n) runtime.

Merge Sort: From Figure 3 and Figure 4, the runtime of Merge Sort is significantly less than that of Selection Sort, Insertion Sort and Bubble Sort. Besides, its growth trend is like a straight line when the input size n is small. This phenomenon confirms our cognition to Merge Sort, whose runtime is always O ($nlog^n$), which is less than O (n^2), the runtime of Selection Sort, Insertion Sort and Bubble Sort.

Quick Sort: In Figure 3, we could clearly see that the line representing Quick Sort grows in the same trend as that of Merge Sort, which means that the runtime of Quick Sort is O ($nlog^n$) for unsorted input arrays. It correctly shows the runtime of Quick Sort with best and average cases, in which T (n) = 2T ($n/2$) + O (n) $\in$ O ($nlog^n$). On the other hand, in Figure 4, the line representing Quick Sort grows in the same trend as that of Selection Sort, which means that the runtime of Quick Sort is O (n^2) for sorted input arrays. It behaves as our expectation because for sorted input arrays, which are worst

cases since our pivot would be the smallest element and one of the partitions is of size n - 1, Quick Sort is effectively Selection Sort, and its runtime is O (n^2).

In conclusion, all the five sorting algorithms behave as expected for both unsorted and sorted arrays.

Question 2

In my opinion, Merge Sort is the best sorting algorithm because its runtime is always O ($nlog^n$), no matter the input array is sorted or unsorted. In contrast, I think Selection Sort is the worst algorithm since its runtime is unchangeable O (n^2) under any circumstances. However, I think the performance of a sorting algorithm depends on the specific occasion in which it is applied.

Question 3

I think it is probable for two reasons:

1. It is difficult to observe the growth trend of lines from the plot with a small input size n since the difference between linear function, quadratic function and log function is not obvious when the value of x-axis is small. With the asymptotically increasing value of input size n, the growth trend would become more obvious, which could tell us the runtime of our algorithm.
2. With a small input size n, the growth trend of runtime is unstable and irregular. It is more likely for the program to generate special cases that would influence our runtime. For example, when we try to figure out the runtime of a sorting algorithm with unsorted input arrays, it is possible for the program randomly generates a sorted input array rather than an unsorted one, and this probability increases with the decrease of n. If the runtime of our algorithm differs from unsorted input arrays and sorted input arrays, then we would get a wrong runtime, which is a data point on our plot, that interferes our verification. With the asymptotically increasing value of input size n, the growth trend of runtime would tend to be stable (most of the input arrays would be general cases) and we could get a more reliable result.

Question 4

The runtime fluctuates for small values of n. As mentioned in Q3, the growth trend of runtime is unstable and irregular for small values of n because of the influence of special cases. For example, we would like to find the runtime of an algorithm with unsorted input arrays while the program randomly provided a sorted one, then the runtime we get differs from the expected answer, which creates a data point away from the trend line on the plot. The probability of this happening is large for small values of n since the program is more likely to generate a special case.

Question 5

We use multiple trials because we would like to weaken the influence from special cases and to get a general conclusion. For example, when we try to figure out the runtime of a sorting algorithm with unsorted input arrays, if we use only one trial, it is possible that the program randomly generates a sorted input array in that trial. If the runtime of our algorithm differs from unsorted input arrays and sorted input arrays, then we would get a completely wrong runtime because the input array is a special case(sorted) rather than a general case(unsorted). In order to get rid of this issue, we would use multiple trials, in which general cases would occupy most of all cases because of the probability of the occurrence of special cases is relatively small, and the result would approach to the expected right answer.

Question 6

It takes more time to run my code when a computationally expensive task is operating in the background. This is because the processor of my computer needs to perform other tasks at the same time, which would delay the speed to run my code.

Question 7

It is because theoretical runtimes would not be influenced by runtime environment while experimental runtimes depend on the hardware and software environments and would vary in different computers. When we would like to verify runtimes of different algorithms, theoretical runtimes provide more useful comparisons. However, when we would like to choose an algorithm to apply to a specified device (i.e., on the server of a company), experimental runtimes provide more useful comparisons.

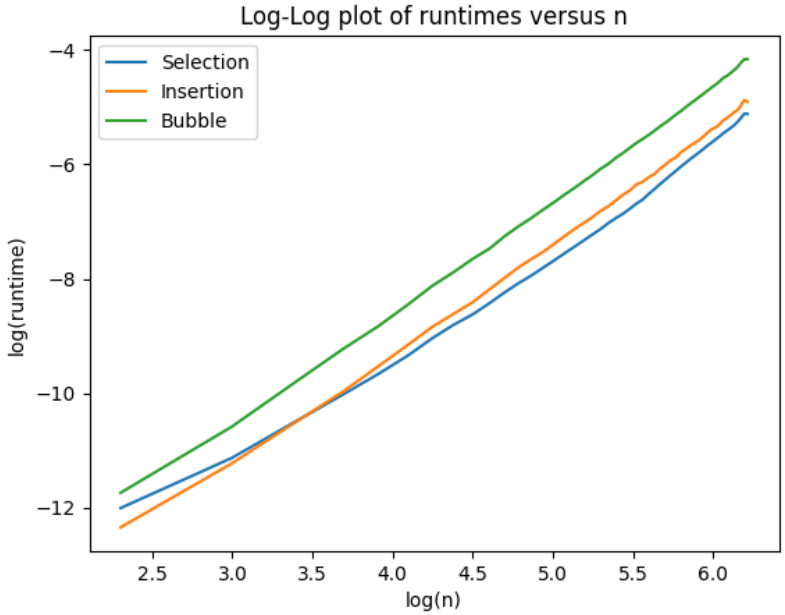

Figure 1. Log-Log plot of sorting runtimes versus input size with randomly generated test arrays

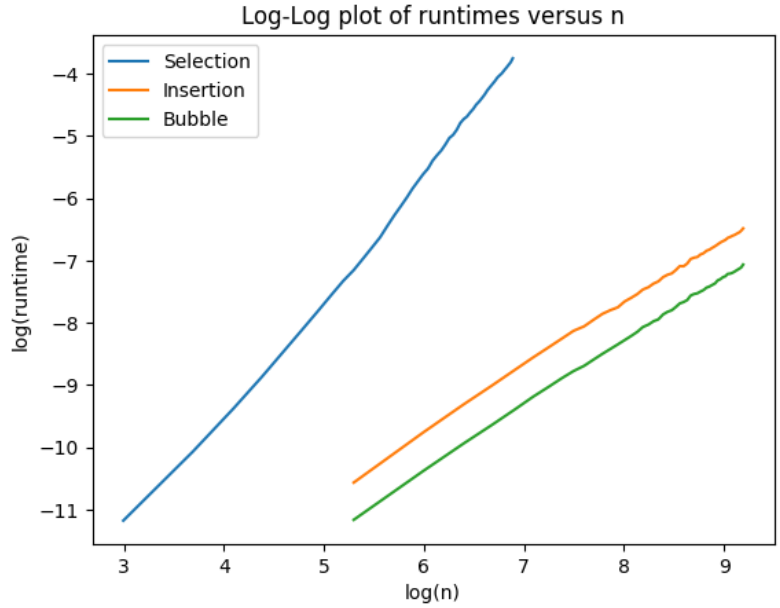

Figure 2. Log-Log plot of sorting runtimes versus input size with sorted test arrays

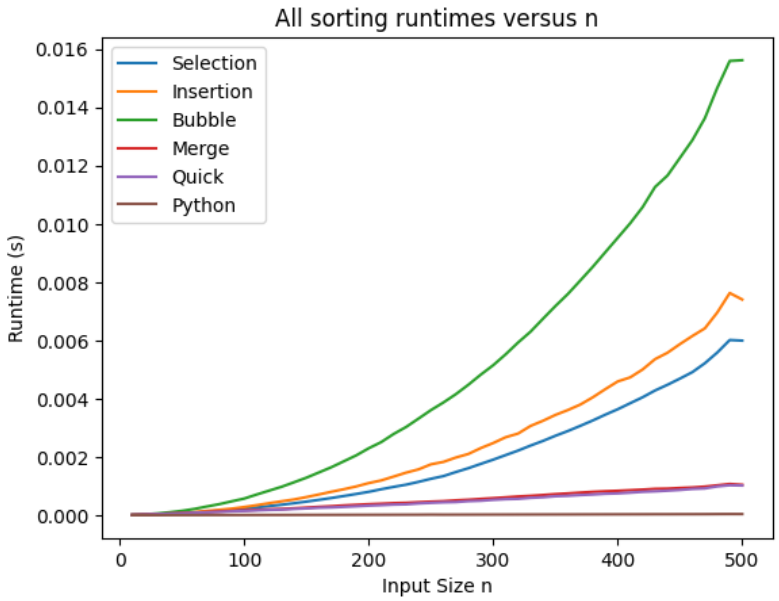

Figure 3. Sorting runtimes versus input size with randomly generated test arrays

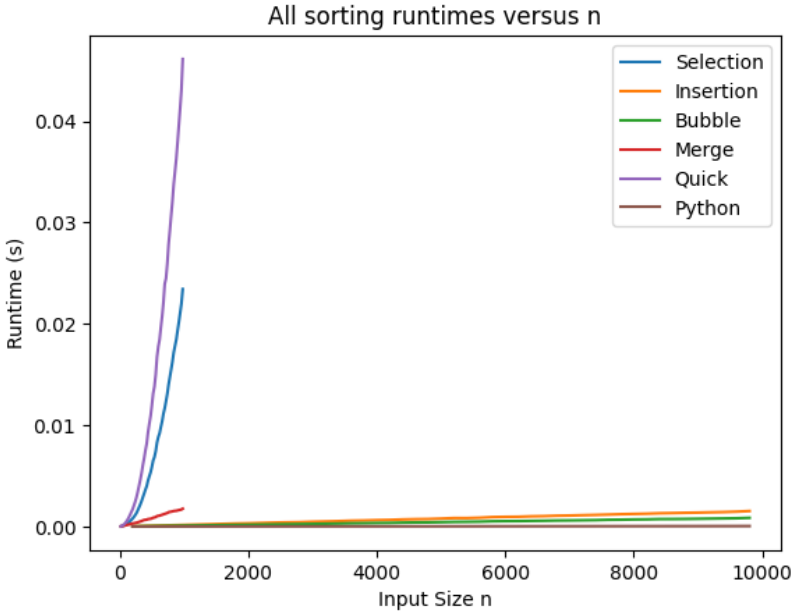

Figure 4. Sorting runtimes versus input size with sorted test arrays