Project One

Weicheng Shi -- ws146

Jiaran Zhou -- jz270

11/01/2018

In this project, we used five sorting methods on unsorted and sorted input arrays. Quick Sort and Merge Sort are much faster than Bubble Sort, Insertion Sort and Selection Sort, and Quick Sort is slightly faster than Merge Sort. The reason is that Quick and Merge have a runtime of O(n*log(n)), but the other three have a runtime of O(n^2). Also, log-log plots of Bubble, Insertion, and Selection Sort are nearly straight lines, which verify the O(n^2) runtime. The behavior of our Sorting methods written in python matched our expectation.

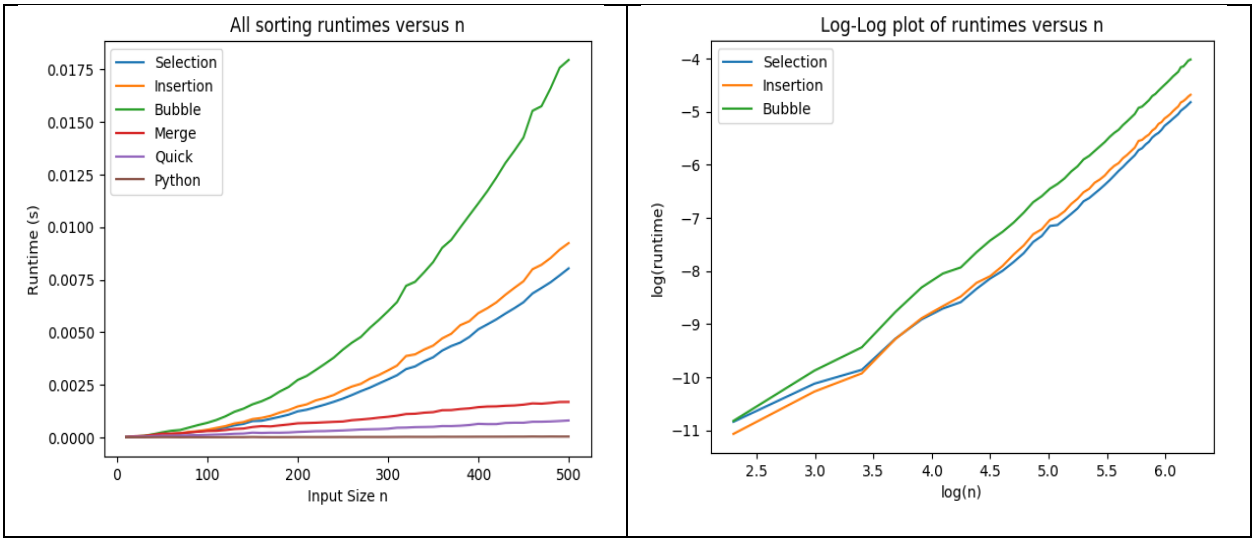

Figure 1. Plots of runtime versus n (unsorted)

When it comes to sorted array, Quick Sort turns out to be the slowest one, as it uses the first element of the array as its pivot. Insertion and Bubble Sort are the best since they have a runtime of O(n) in this situation. Our result matched the desired behavior as well.

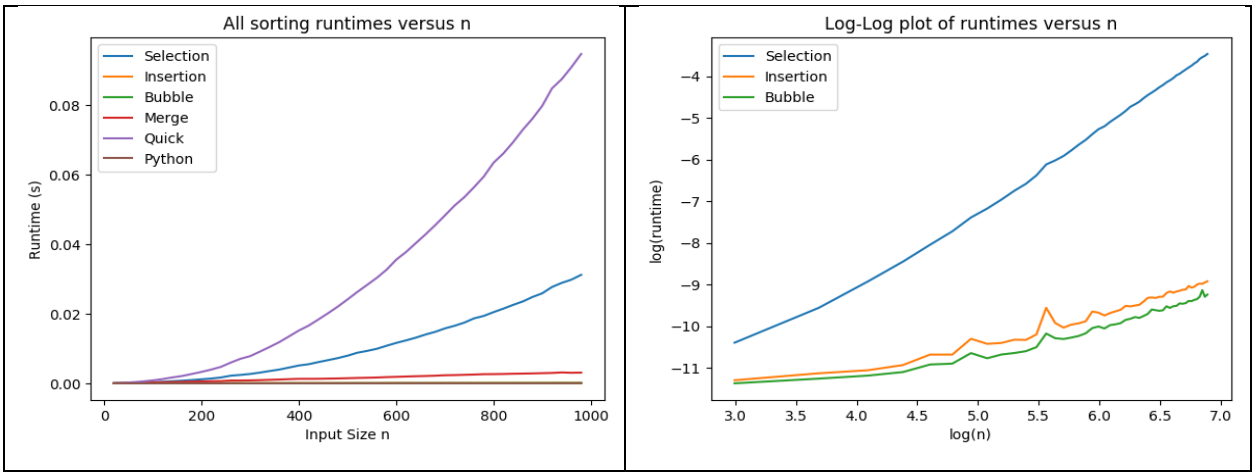

Figure 2. Plots of runtime versus n (sorted)

Among these five methods, Merge Sort and Quick Sort are better. To compare these two, we need to take different conditions into consideration. For worst cases, runtime of Quick Sort is O(n^2), while runtime of Merge Sort stays O(n*log(n)). As a result, for those jobs which require stable computation speed such as rocket launch, we can only use Merge Sort. On the other hand, average runtime of Quick Sort is faster than that of Merge Sort, so it is better to use Quick Sort in people's daily life. Also, Quick Sort does not need to allocate new space during its operation. Bubble Sort is the worst algorithm. Though it shares similar runtime of O(n^2) with Insertion Sort and Selection Sort, its runtime time is much slower than that of the other two methods.

The reason why large values of n should be reported is that computer programs are typically used for a large amount of input data. Also, the Big-O Theory only works for asymptotically large n. However, behaviors of small input size may be different. For example, when we have two methods with runtime n^2 and 1000000n, it is faster for the first method when n is small, and the first one is slower when n is large.

Small amounts of trails introduce random behaviors, which cannot be used to conclude general behavior of a method. According to the runtime vs n graph with one trail, plot oscillates drastically, and the log-log plots are no longer straight lines. These behaviors do not meet our expectation. When we compare graphs with one trail and 30 trials, the one with more trials is much smoother than the other. With more trials, random behaviors occur much less frequently, So, we always average the runtime across multiple trials as it can better represent the general behaviors of an algorithm.

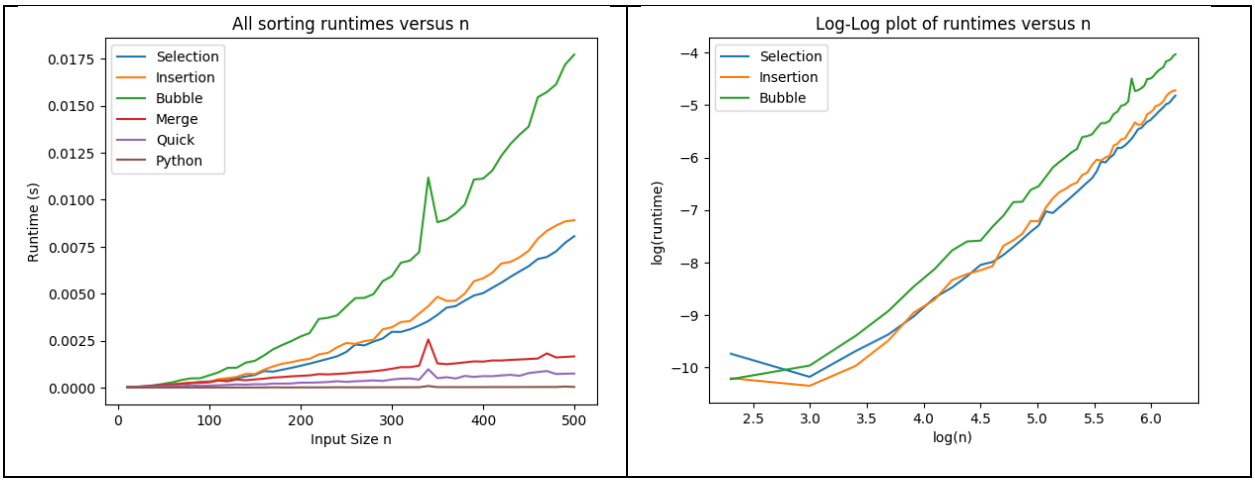

Figure 3: runtime plot for one trail

We tried running several programs in the background when we tested our sorting algorithms. We realized that the plot of runtime oscillated significantly. To be specific, it rose when we ran multiple programs at the same time. When we want to test performance of algorithms, it is better to test with no background programs running.

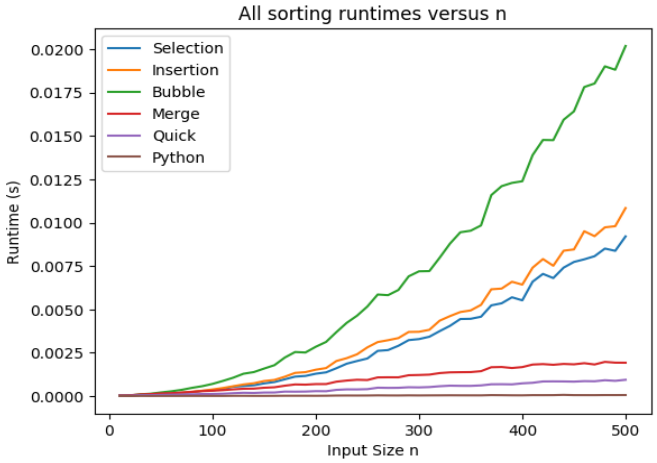

Figure 4: runtime plot with background programs running

Different machines and conditions may influence actual runtime of algorithms. When we compare performance of different algorithms, it is better to report the theoretical runtime using Big-O method. However, there are many restrictions that may influence the actual performance of algorithms, such as temperature and computers with different computing power. So, in industry, we sometimes compare experimental runtime under different conditions. For example, when the actual runtime does not meet our needs, other actions such as more cooling power may help improving computation performance.