Team Members: Casper Hsiao and Leon Zhang

ECE 590 Project 1 Report

Do your algorithms behave as expected for both unsorted and sorted input arrays?

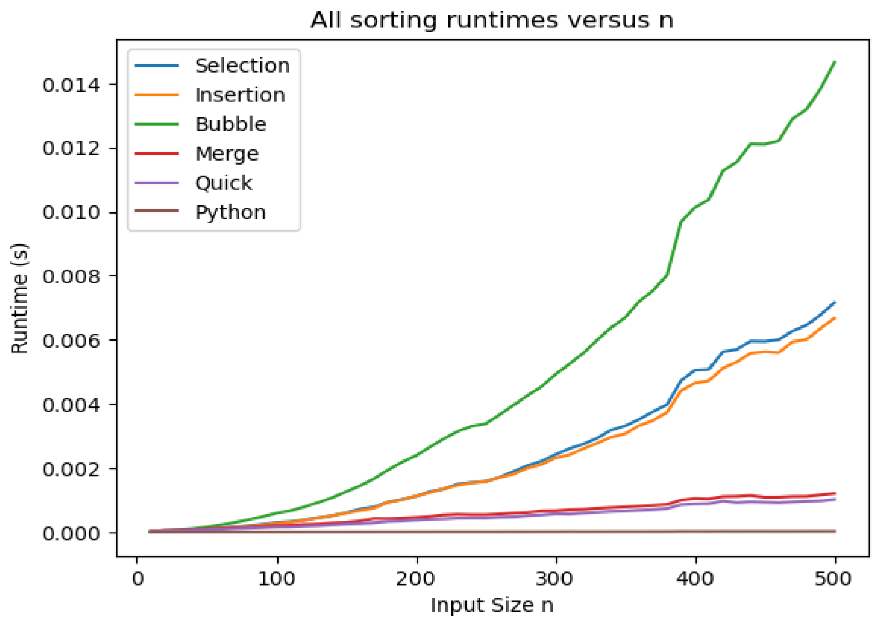


Figure 1

All the algorithms behaved as expected. As bubble sort, selection sort and insertion sort runtime grew in O(N^2) fashion and merge sort, quick sort fell into the O(NlogN) category.

Which sorting algorithm was the best (in your opinion)? Which was the worst? Why do you think that is?

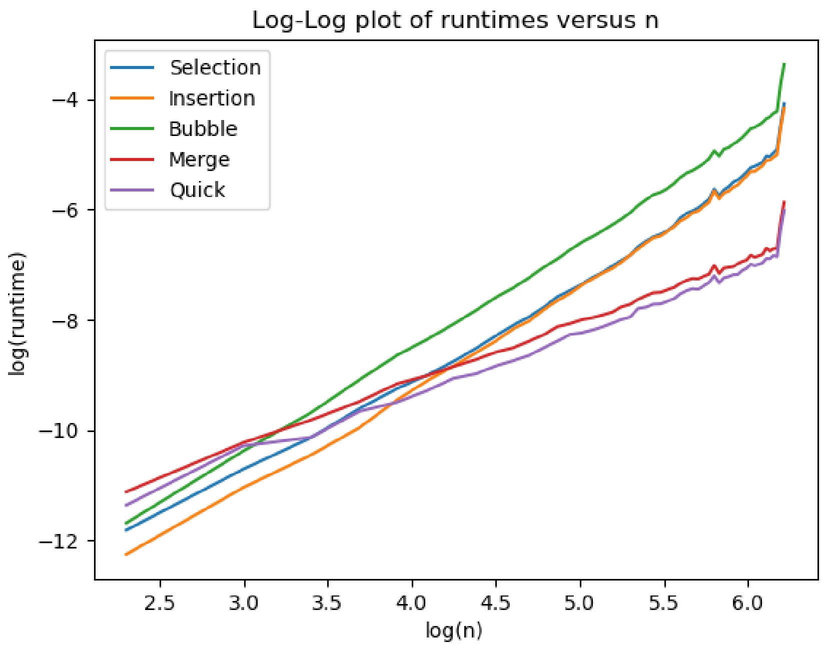
Quick sort, followed by merge sort were the best performing algorithms. They both run in average O(NlogN) time. Quick sort runs in average O(NlogN) time because

Why do we report theoretical runtimes for asymptotically large values of n?

With regards to asymptotically large values of n, measuring actual runtime becomes impractical, since we would not be able to measure the actual time when the input is asymptotically large.

Team Members: Casper Hsiao and Leon Zhang

What happens to the runtime for smaller values of n? Why do you think this is?



The figure above shows that the slope for smaller values of n is relatively “unfitted” compared to runtime for larger values of n. To analyze this phenomenon, we need to understand that Big-O accounts for large values of n and ignores the constant operations, since it is insignificant in cases of large n values and the n factor dominates the run time. However, for small values of n, the constant operation actually is much more significant, since the n factor is smaller and does not dominate the run time. This is why the slope for smaller values of n is relatively theoretically inaccurate.

Why do we average the runtime across multiple trials? What happens if you use only one trial?

First of all, actual runtime is highly dependent on the machine the algorithm runs on. In fact, actual runtime even differs if it is run on the same machine multiple times. This fact makes the actual runtime unreliable as a standard benchmark when measuring the algorithm time complexity. Therefore, to maximize the effectiveness of actual run time, we take the average across multiple trials to produce a rather reliable run time.

What happens if you time your code while performing a computationally expensive task in the background (i.e., opening an internet browser during execution)?

The computationally expensive tasks in the background will definitely affect the run time of the code. This is because the tasks will use a high percentage of the CPU to compute its complex processes, therefore, affecting the CPU to allocate resources to run the code fully.

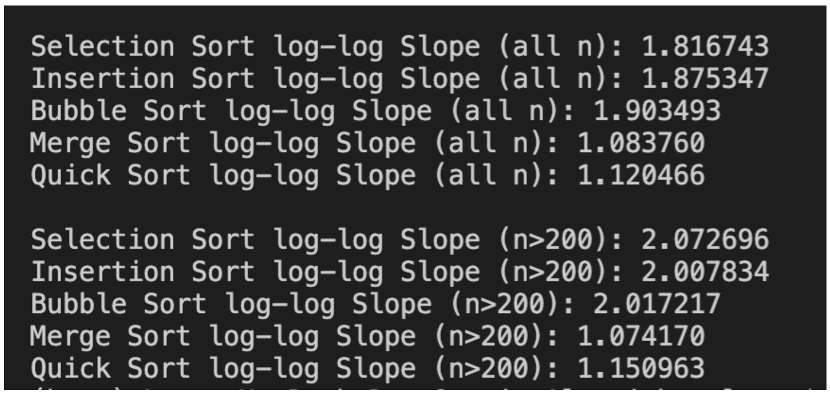
Team Members: Casper Hsiao and Leon Zhang

Why do we analyze theoretical runtimes for algorithms instead of implementing them and reporting actual/experimental runtimes? Are there times when theoretical run-times provide more useful comparisons? Are there times when experimental runtimes provide more useful comparisons?

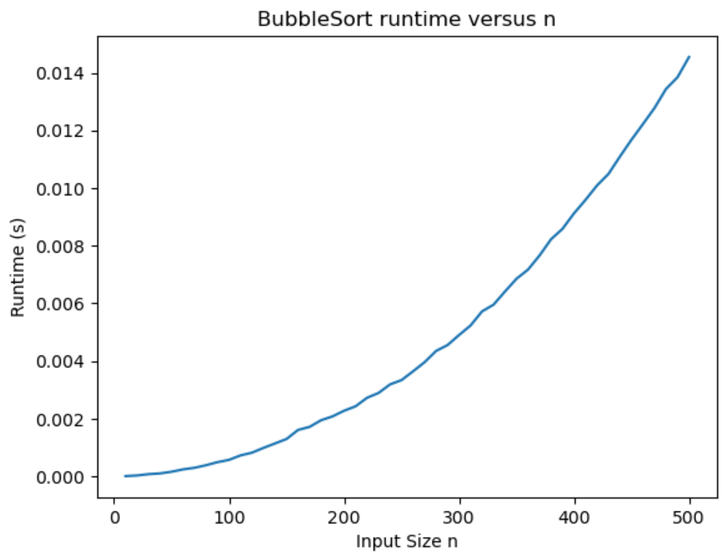
We would generally prefer theoretical run times since it is more of a standard benchmark. As mentioned previously, the actual runtime depends highly on the machine you run on. For instance, it is possible that bogo sort runs faster on a supercomputer than quick sort that ran on a computer in the 1990s. However, this does not prove that bogo sort is “actually” faster than quick sort. When comparing the time complexity of different algorithms, we should use theoretical runtime to generate accurate comparisons. In some cases, when we want to optimize the runtime of an algorithm on a certain hardware, we would then consider the actual runtime. For instance, if we want to optimize the autonomous driving algorithm of a Tesla car, we would want the algorithm to fully complement its hardware. In this case, measuring algorithms with actual runtime would be useful.

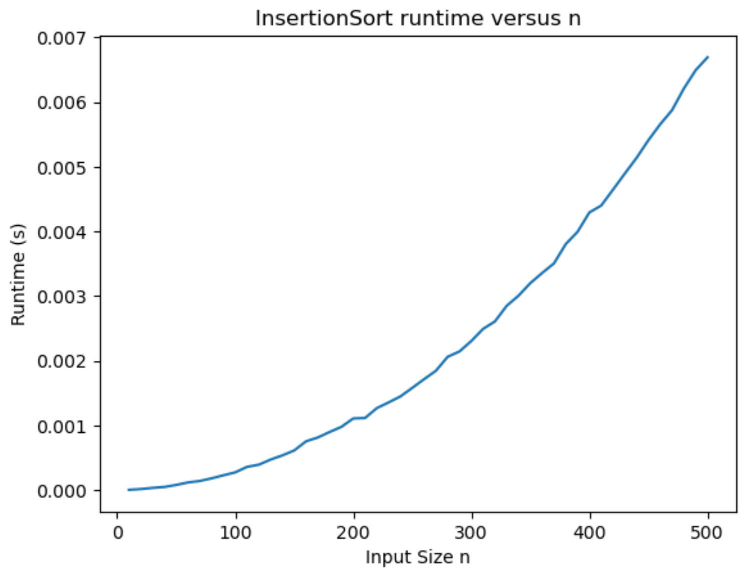
Team Members: Casper Hsiao and Leon Zhang

Appendices - Graphs

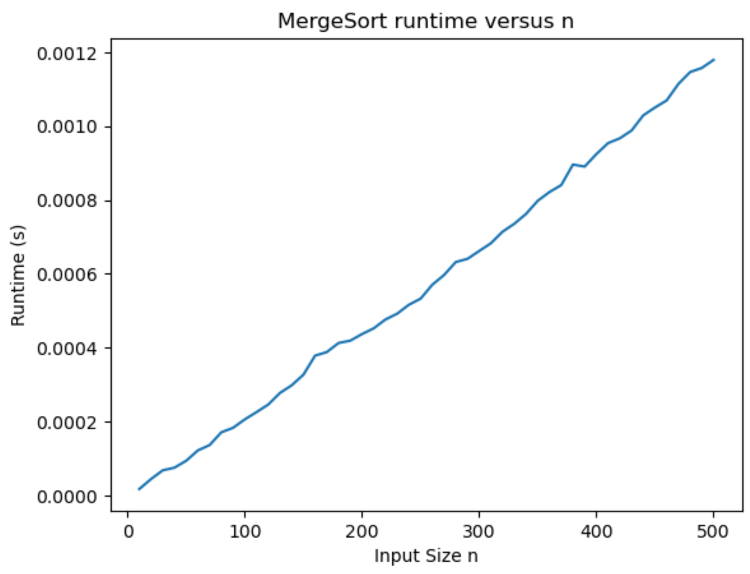


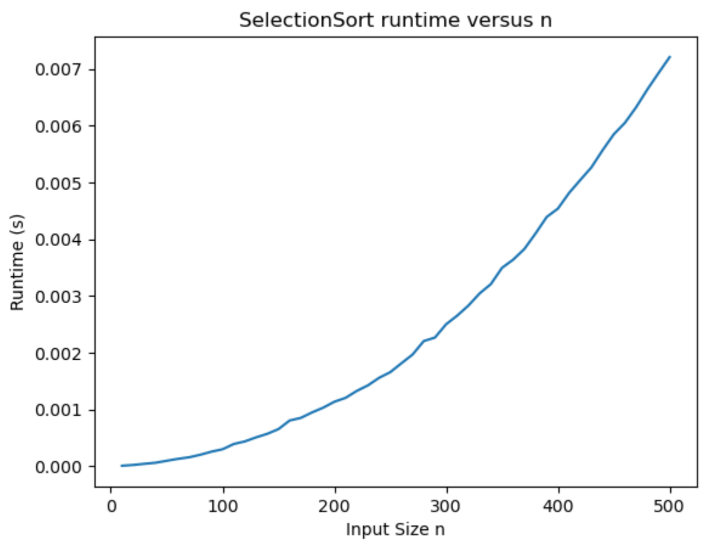
```
Selection Sort log-log Slope (all n): 1.816743
Insertion Sort log-log Slope (all n): 1.875347
Bubble Sort log-log Slope (all n): 1.903493
Merge Sort log-log Slope (all n): 1.083760
Quick Sort log-log Slope (all n): 1.120466

Selection Sort log-log Slope (n>200): 2.072696
Insertion Sort log-log Slope (n>200): 2.007834
Bubble Sort log-log Slope (n>200): 2.017217
Merge Sort log-log Slope (n>200): 1.074170
Quick Sort log-log Slope (n>200): 1.150963
```

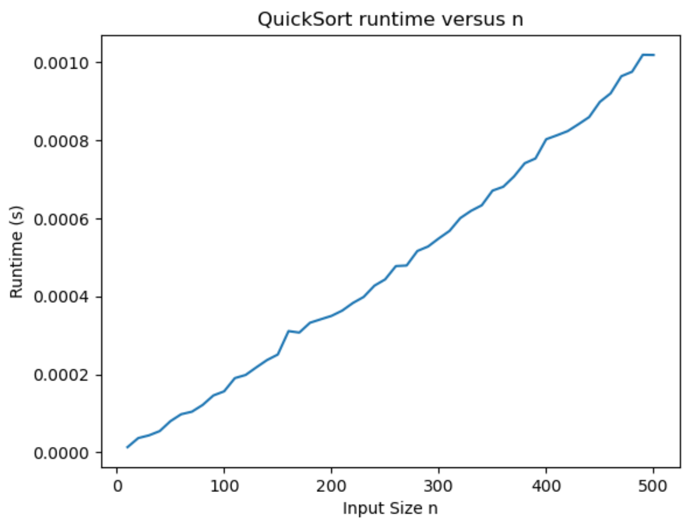




Team Members: Casper Hsiao and Leon Zhang


MergeSort runtime versus n
Runtime (s)
Input Size n


SelectionSort runtime versus n
Runtime (s)
Input Size n


QuickSort runtime versus n
Runtime (s)
Input Size n