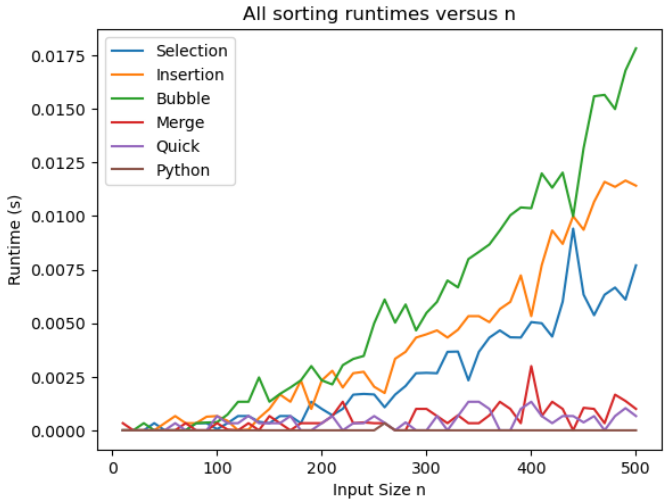
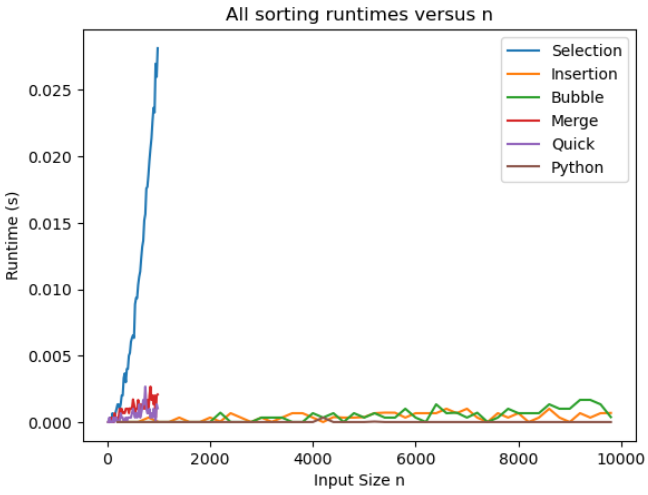
ECE590/MATH560 Project 1 Report

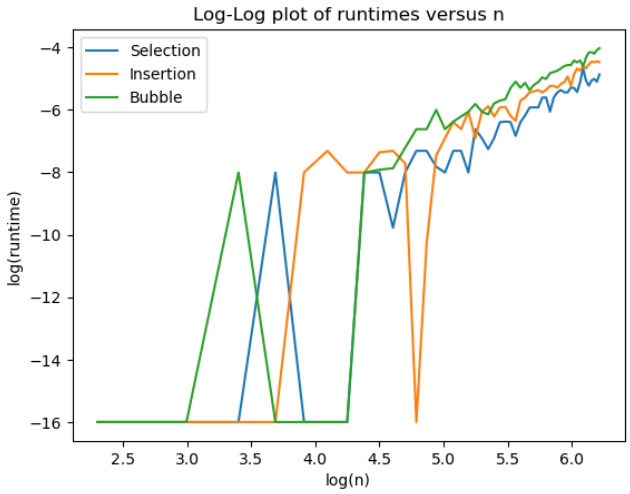
Libo Zhang, NetID: lz200

Question 1 – Do your algorithms behave as expected for both unsorted and sorted input arrays?

Answer: Yes. However, the results seem to be very noisy. Below are some figures acquired after running the main function in project1.py.








```
UNSORTED measureTime

Timing algorithms using random data.
Averaging over 30 Trials


Selection Sort log-log Slope (all n): 3.342645
Insertion Sort log-log Slope (all n): 3.166825
Bubble Sort log-log Slope (all n): 3.545736

Selection Sort log-log Slope (n>200): 2.365946
Insertion Sort log-log Slope (n>200): 2.023374
Bubble Sort log-log Slope (n>200): 2.090620
Merge Sort log-log Slope (n>200): 1.526308
Quick Sort log-log Slope (n>200): 2.276797

SORTED measureTime

Timing algorithms using only sorted data.


Selection Sort log-log Slope (all n): 2.968586
Insertion Sort log-log Slope (all n): 2.157745
Bubble Sort log-log Slope (all n): 3.341189

Selection Sort log-log Slope (n>400): 2.110131
Insertion Sort log-log Slope (n>400): -0.267130
Bubble Sort log-log Slope (n>400): 2.282603
Merge Sort log-log Slope (n>400): 0.551493
Quick Sort log-log Slope (n>400): -0.516914

C:\Users\Administrator\Desktop\590Libo>
```

For unsorted arrays, the top left figure shows the runtime of five sorting algorithms. We can notice that bubble sort, insertion sort, and selection sort show a runtime of $O(n^2)$, while merge sort and quick sort show a runtime of $O(nlogn)$, considering the average case for all sorting algorithms.

For sorted arrays, which means the best-case situation, the top right figure shows that selection sort has a runtime of $O(n^2)$, merge sort and quick sort have the runtime of $O(nlogn)$, while insertion sort and bubble sort have the runtime of $O(n)$. Therefore, we can see from the top 2 figures that, my algorithms behave as expected for both unsorted and sorted arrays.

However, according to the bottom two figures, we can find that the running results are very noisy. Consider the bottom left figure, only when n is large enough, we can see that there is a straight line with a slope of 2 for selection, insertion and bubble sort. And for the bottom right figure, I even get negative values for log-log slope when n is greater than 400! After seeking advice from the professor during his Office Hour, my analysis is that when n is relatively small, the array size is so small that the sorting runtime of my computer is very short. The runtime is so short that my computer could not precisely record the time that Python spent in sorting. The runtime is so short, and the noise is so large, therefore, the magnitudes of my slope values are to some extent distorted.

Question 2 – Which sorting algorithm was the best (in your opinion)? Which was the worst? Why do you think that is?

Answer: Based on the running results, I think merge sort is the best sorting algorithm, because it has the runtime of $O(nlogn)$ for all best, average, and worst cases, although insertion and bubble sort have $O(n)$ runtime under the best case. Since we do not always sort an array that has already been sorted, I think merge sort is the best algorithm considering the tradeoff of best/average/worst case.

I think the worst algorithm is selection sort, because it has the longest runtime $O(n^2)$ under all cases.

Question 3 – Why do we report theoretical runtimes for asymptotically large values of n?

Answer: First, when we work on real-life projects, the dataset could be very large, which means n could be very large, so it is important for us to estimate theoretical runtime and try to save computational power. Second, different computers have different hardware and different computing abilities, that is why we need to report runtime in an asymptotical way.

Question 4 – What happens to the runtime for smaller values of n? Why do you think this is?

Answer: As mentioned in Question 1, small values of n mean the runtime of Python sorting will also be very small. The runtime could be so small that my computer Windows 10 operating system may fail to precisely record such a small runtime, and the timing results could be very noisy or totally in a mess.

Question 5 – Why do we average the runtime across multiple trials? What happens if you use only one trial?

Answer: We average the runtime across multiple trials because we want to acquire more stable and more convincing results. If we only use one trial, such one-trial experiment may give us very noisy or very random results, and therefore making the experimental results less reliable and less convincing.

Question 6 – What happens if you time your code while performing a computationally expensive task in the background (i.e., opening an internet browser during execution)?

Answer: This will influence the performance of the timer of my computer's Windows 10 operating system. Because we know that one task is for Python to do sorting, and the other task is opening an internet browser, but the operating timer does not know such difference. Therefore, the timing for sorting execution will be influenced by such a computationally expensive task in the background, and the results could be very noisy, even if we have relatively large values of n.

Question 7 – Why do we analyze theoretical runtimes for algorithms instead of implementing them and reporting actual/experimental runtimes? Are there times when theoretical runtimes provide more useful comparisons? Are there times when experimental runtimes provide more useful comparisons?

Answer: Because different computers have different levels of hardware, which means that different computers have different computing power/resources/abilities. For the same task, a supercomputer may only take 30 minutes to complete, while a normal computer may take years to complete this "simple" task! Therefore, experimental runtimes could not give us a standard/unified way of measuring the complexity of one specific algorithm/solution. If we are working on a project with the help of one specified computer, since we have known the computing power of the computer, when we choose which algorithm to use, this is the time when theoretical runtimes providing more useful comparisons, and we would choose the one that has the smallest runtime.

However, consider another situation, we have one CPU, one GPU, and one TPU simultaneously, and we know that our algorithm/model/solution performs differently on different hardware platforms. This is the time when experimental runtimes provide more useful comparisons.