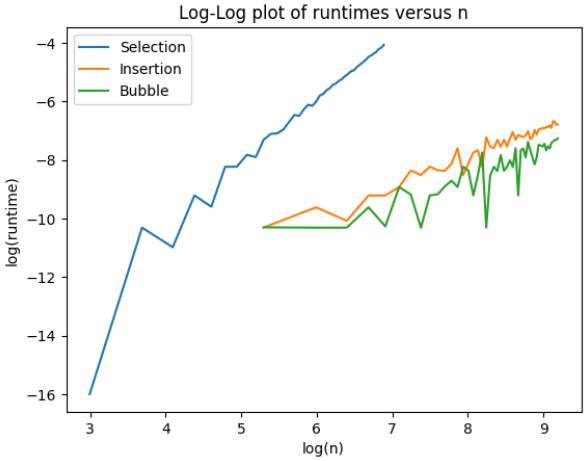
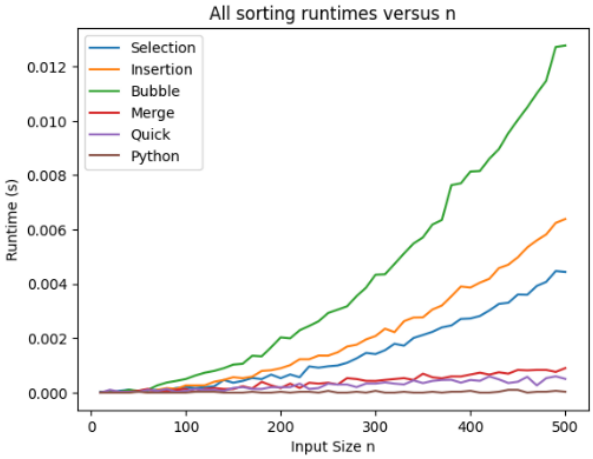
Ching-Hang Hsu ch450

Zhuoer Li zl328

1. Yes, in the figures below, we can observe that all the algorithms behave as expected, and we list each runtime performance in the following table. In particular, bubble and insertion sort are with runtime O(n) in sorted cases, but with $O(n^2)$ in unsorted cases because the slope of the Log-Log plot is closed to 2.






```
UNSORTED measureTime

Timing algorithms using random data.
Averaging over 30 Trials


Selection Sort log-log Slope (all n): 1.888904
Insertion Sort log-log Slope (all n): 3.205263
Bubble Sort log-log Slope (all n): 2.291375

Selection Sort log-log Slope (n>200): 2.254580
Insertion Sort log-log Slope (n>200): 2.119276
Bubble Sort log-log Slope (n>200): 2.112033
Merge Sort log-log Slope (n>200): 1.448478
Quick Sort log-log Slope (n>200): 1.071834

SORTED measureTime

Timing algorithms using only sorted data.


Selection Sort log-log Slope (all n): 2.431899
Insertion Sort log-log Slope (all n): 0.985045
Bubble Sort log-log Slope (all n): 0.927821

Selection Sort log-log Slope (n>400): 2.061148
Insertion Sort log-log Slope (n>400): 0.900614
Bubble Sort log-log Slope (n>400): 1.222670
Merge Sort log-log Slope (n>400): 1.108484
Quick Sort log-log Slope (n>400): 2.061553
jerry@DESKTOP-K3NKS0L ~\OneDrive - Duke University\Duke\590\project\project1
```

Algorithms	**Unsorted**	**Sorted**
Selection sort	$O(n^2)$	$O(n^2)$
Insertion sort	$O(n^2)$	$O(n)$
Bubble sort	$O(n^2)$	$O(n)$
Merge sort	$O(n \log n)$	$O(n \log n)$
Quick sort	$O(n \log n)$ / $O(n^2)$	$O(n \log n)$

2. Since it's rare to have a sorted array in practice, we think that focusing on the performance of unsorted cases is more practical. Therefore, according to the figure above, the best sorting algorithm we thought is merge sort and the worst is bubble sort.
3. (3&4) Since the cases with large values of n are more randomized (i.e., the permutation of n is more complex), the theoretical runtimes are more approached to the asymptotic line. On the other hand, the cases with smaller values of n have the simpler permutations, so it's easier to get a sorted array and have less operation in sorting. As a result, the theoretical runtimes for asymptotically small values of n are smaller than the runtimes in large cases.
5. To calibrate the theoretically runtimes of each algorithm, we need to test several cases to get an average runtime. Since we can't promise that every case is random enough (i.e., we might get a sorted array accidentally for the unsorted case), averaging the runtime across several trials could give us a more precise result.
6. We can observe that there are some oscillations in the runtime while performing a computationally expensive task in the background. We think the reason is that the CPU will become unstable while running multiple tasks at the same time.
7. From the above questions we can know there're multiple factors that can affect our experimental results, such as multitasking of CPU and not random enough arrays(or even unsorted). With the theoretically runtimes, we can compare our experimental results to verify the correctness and find if there're any errors. On the other hand, sometimes, we can't differentiate two algorithms with the theoretical runtimes. However, through the experiments, we can easily observe the differences. For instance, selection sort and bubble sort with the same theoretical runtime $O(n^2)$, but with different slopes in experimental results. To conclude, to get a better analysis, we need both theoretical and experimental runtimes of each algorithm.