Project Report 1: Sorting

Students Name and NetID:

- Yifan Men(ym129)
- Yuqiao Liang(yl543)

Selection Sort

The main idea of selection sort that we have two array, one is sorted and the other is unsorted. We pick up the smallest element from the unsorted array and add it to the end of the sorted array.

Insertion Sort

The idea behind insertion sort is similar to selection sort. The difference is that in insertion sort, we pick the next element from the unsorted list instead of the smallest. Then we insert this element into the sorted array at the correct order(by ascending order).

Bubble Sort

This algorithm requires iterations through the array. By comparing the pairs of numbers, it makes the the pairs in an ascending order. Keep doing so until no more swaps are made. For each pass, the kth largest element gets placed at the kth iteration. So we only need to iterate through the first $n - k$ elements at each pass.

Merge Sort

It is a divide and conquer algorithm. This algorithm try to divide the array into two halves, so that the small part will be sorted. The base case of each subarray is 1–element or 2–element array. With two sorted subarray, it is easy to merge the two with a new array which is sorted.

Quick Sort

Quick sort use the divide and conquer philosophy. First we need to choose one element as pivot. Then partition the array into two parts. On the left is everything smaller than pivot, on the right is everything bigger than pivot. After each partition, we will recursively call the partition process.

Overall

Time Complexity

Algorithm	Best Case	Average Case	Worst Case
Selection Sort	$O(n^2)$	$O(n^2)$	$O(n^2)$
Insertion Sort	$O(n)$	$O(n^2)$	$O(n^2)$
Bubble Sort	$O(n)$	$O(n^2)$	$O(n^2)$
Merge Sort	$O(nlogn)$	$O(nlogn)$	$O(nlogn)$
Quick Sort	$O(nlogn)$	$O(nlogn)$	$O(n^2)$

Testing Methodology

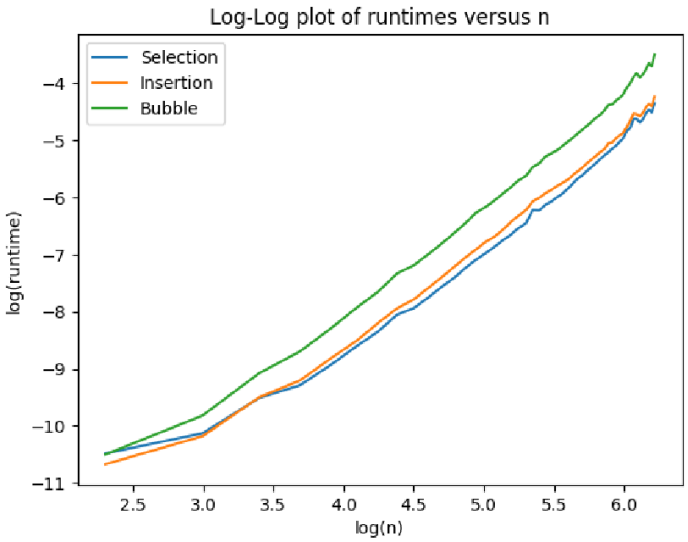
We do multiple test, and calculate the average value as the answer. In such a way, we can minimize environment–caused discrepancy. We take 3, 30, 120 test trials, though might not big enough with the real world data.

As the performance of each computer is different, it's reasonable to report the theoretical runtime. Once we calculate the expected runtime, we can compare it with the runtime we obtain to see if our algorithm is implemented correctly. Runtime for smaller values of n is not convincing. If we were running another application in the background, which will slow down the computer, we can have huge difference between the idle condition and the busy condition.

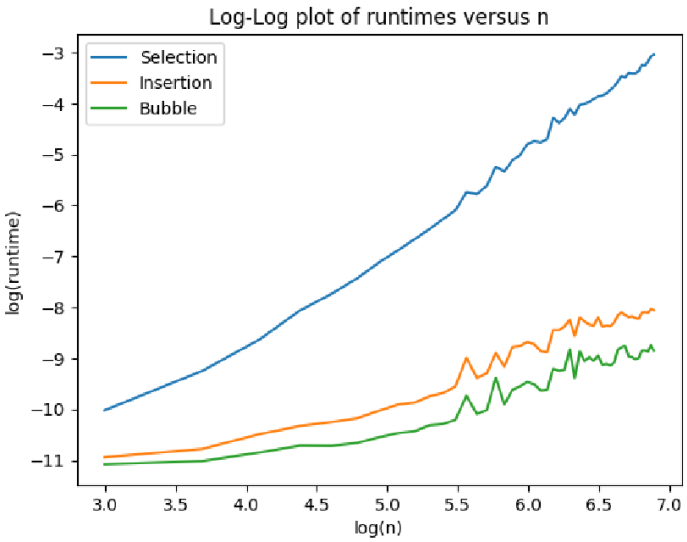
Realistically, the error of each algorithm cannot be avoided. Actual tuntimes may be influenced by lots of other factors as we mentioned before. When we are dealing with large amounts of data, we ought to rely more on theoretical runtimes, since it's

very inefficient to simulate each algorithm. However, when we are dealing with small data or data has some restriction, it's better to run each algorithm and pick the best one.

Best Case comparison



Unsorted array with 30 trials



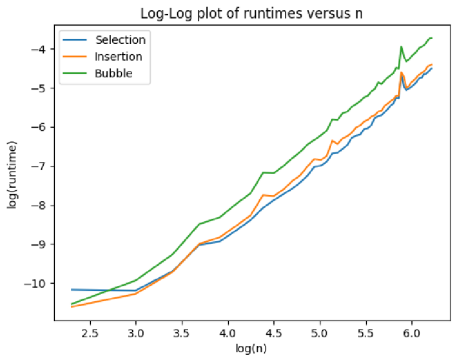
Sorted array with 30 trials

According to the left figure, all of Selection Sort, Insertion Sort, Bubble Sort run in the same big O of complexity(i.e. $O(n^2)$), when the array is not sorted. However, when it comes to a sorted array(the right figure), Selection Sort behaves the different way, this is because the other two has some optimization to this situation, avoiding useless iteration.

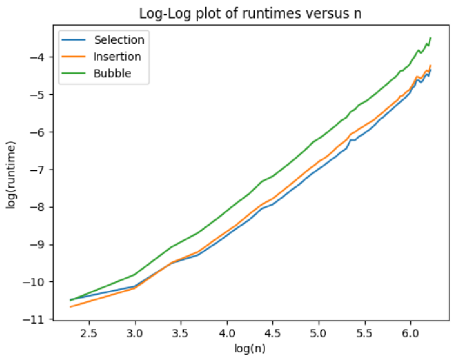
What's more, the reason why bubble sort usually takes more time than the other two may because the swap takes more time, even it is $O(1)$ time complexity for each swap.

Best Sorting Algorithm

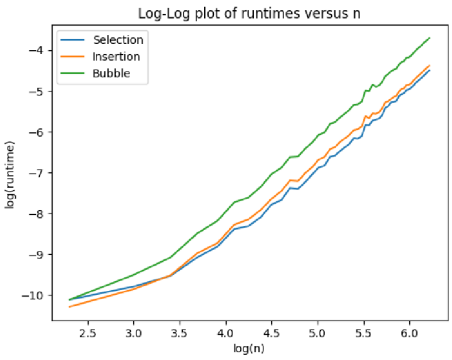
Here we have several algorithms: Selection Sort, Insertion Sort, Bubble Sort, Merge Sort, Quick Sort, Python Built-in Sort. We compare these algorithms with 3-trials array(the left figure), 30-trials array(the center figure) and 120-trials array(the right figure).



Unsorted array with 3 trials



Unsorted array with 30 trials



Unsorted array with 120 trials

All of them behaves the same way. Although the worst case of Quick Sort is also $O(n^2)$, but it does not really happen in real situation, and it is also the fastest algorithm according to tests.

When we take the Python Built-in Sorting into consideration, there is no-doubt that it is always the best choice. According to Internet, it is a method called Timsort, which is a hybrid sorting algorithm, derived from merge sort and insertion sort, designed to perform well on many kinds of real-world data.