VE281 Project 1
Weikai Zhou 518021911039

1. Performance of Small Cases

We first studied the performance of small cases, which I choose is between 5 numbers to 25 numbers. For each case, I run 5 times to get the average value. The time needed for each case is shown in the figure below (Figure 1).

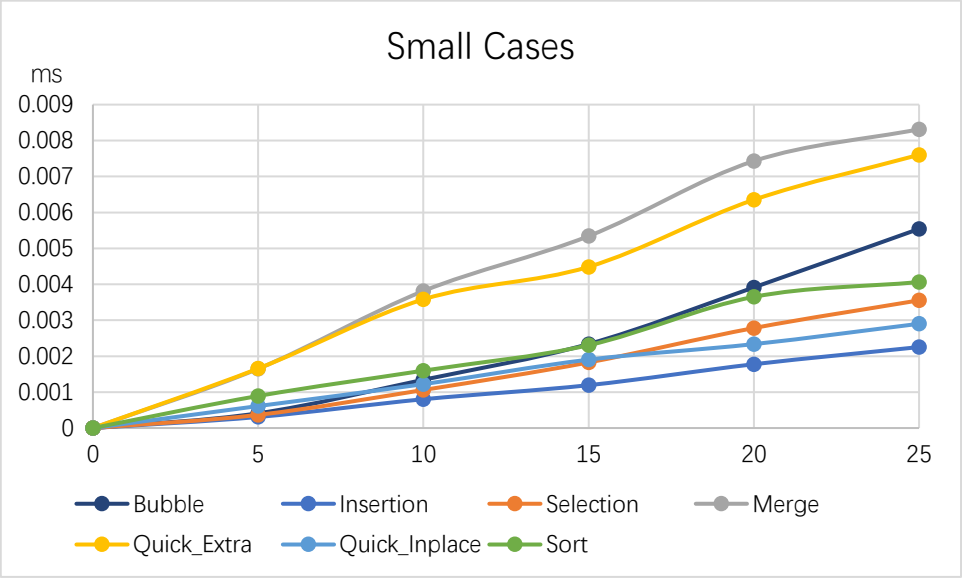

Figure 1. Small cases

From the figure, we can find that `MergeSort` and `QuickSortExtra` are significantly slower than other sorts, while other sorts are almost in the same level especially when the case is exceedingly small, e.g. 5 numbers to sort. Generally, `std::sort` is based on `QuickSort`, and the time complexity is *O(nlogn)*. But there may be a threshold such that when *n* (the number of values) is small than the threshold, it will adopt `InsertionSort` so that it can be faster. Because for small cases, some pieces of values are in order and we can achieve *O(n)*. While for `QuickSort`, compared with `InsertionSort`, partition is time consuming in small cases.

2. Performance of General Cases

We then studied the performance of general cases, which I choose is between 10^2 numbers to 10^7 numbers. For each case, I also run 5 times to get the average value. The time needed for each case is shown in the figure below (Figure 2).

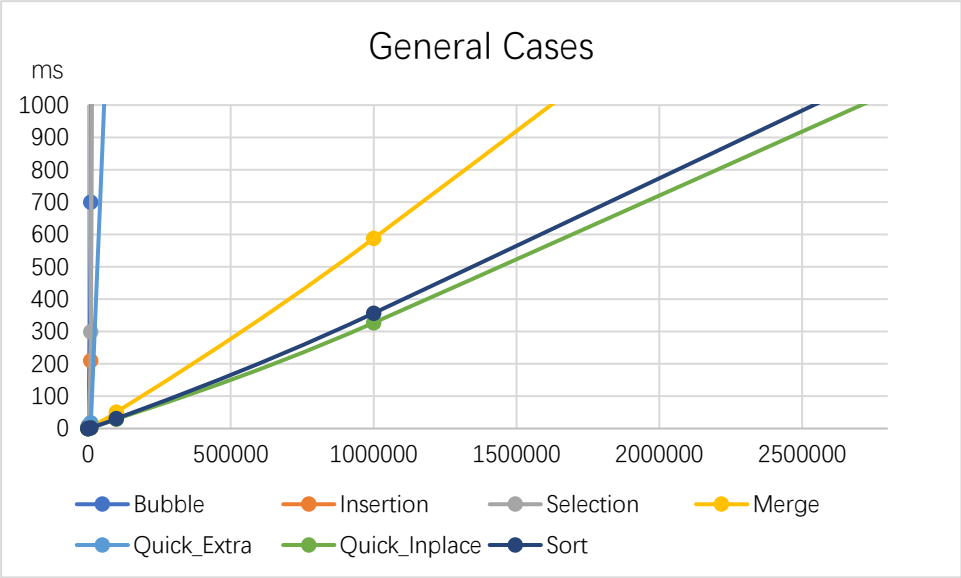

Figure 2.a. General cases in large scale

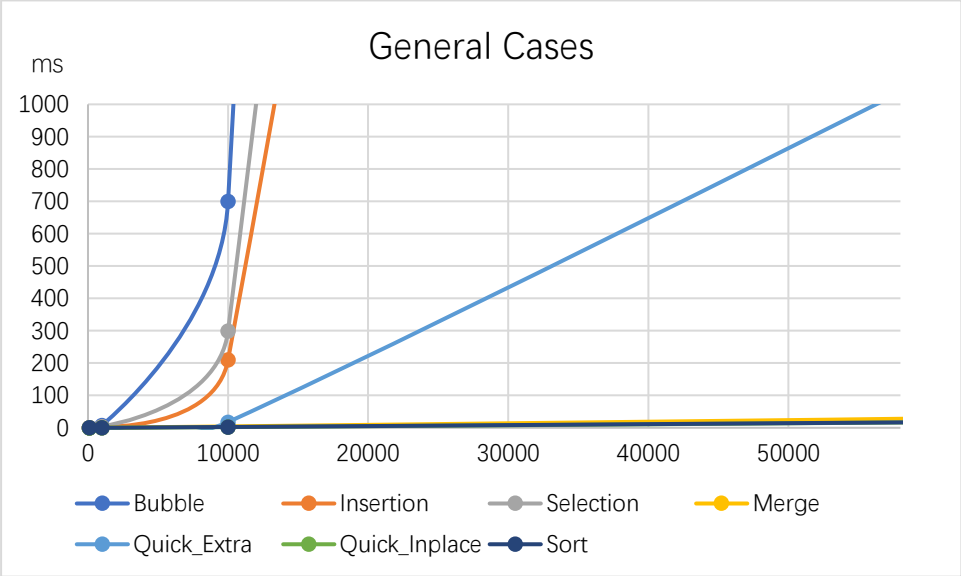

Figure 2.b. General cases in detailed scale

From the figure, we know the time needed for `BubbleSort`, `InsertionSort` and `SelectionSort` grows quickly as n increases, while for `MergeSort`, `QuickSort` and `Sort`, they grow slowly. This is because the formers are in $O(n^2)$ while the latter are in $O(nlogn)$. But `QuickSortExtra` also grows quickly and is even close to $O(n^2)$ in Figure 2.a. This is due to the reason that it uses extra space and copying elements between the extra and the original space is time consuming. Also, `MergeSort` is a little bit slower than `QuickSortInplace` and `Sort` because it is not in place.