VE281 Project 1

Mingxuan Lu 518021911166

1. Theoretical Data

As discussed in Chapter 2, the time complexity for each sorting algorithms is listed in the table follow.

	Worst Case Time	Average Case Time	In Place	Stable
Insertion	$O(N^2)$	$O(N^2)$	Yes	Yes
Selection	$O(N^2)$	$O(N^2)$	Yes	No
Bubble	$O(N^2)$	$O(N^2)$	Yes	Yes
Merge Sort	$O(NlogN)$	$O(NlogN)$	No	Yes
Quick Sort	$O(N^2)$	$O(NlogN)$	Weakly	No

2. Test Result

2.1 Test With Different Array Size

In this section, arrays with different size are generated to test all seven, std::sort included, sorting algorithms' performance with arrays with size from 1 to 10000. In the test program, C++11 chrono library is used to get more accurate runtime of functions than std::clock. The result is listed in the table below (Table 1).

Array Size	Bubble Sort	Insertion Sort	Selection Sort	Merge Sort	Quick Sort Extra	Quick Sort In-place	std::sort
1	0.6	0.8	0.7	1	0.8	0.7	0.7
10	1.6	1.7	1.5	10.1	7.5	2.1	2.5
100	77.8	30.1	55.8	55.7	36.7	17.8	22.9
500	2034.1	863.6	2415.5	365.3	240.6	117.5	119.4
1000	8272.7	2274.2	4358.9	599.5	458.2	344.2	242.3
5000	243731	92248.6	102224	3529.2	3826.1	1478.3	1637.1
10000	1082240	248294	369619	7138.5	6842.4	4081.5	4368.8

Table 1 Seven Sorting Algorithms' Performance With Arrays of Different Size

The performance of all seven sorting algorithms is plotted using Matlab in the figure below (Figure 1).

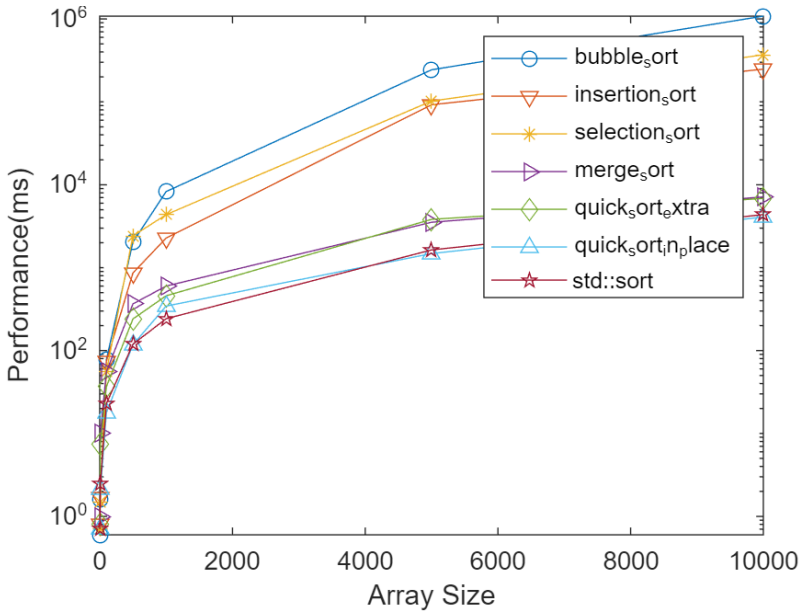

Figure 1 Run Time Comparison

As a result, a few conclusions can be made.

1. For each algorithm, run time increases when the size of the array increases.
2. When the array size is small (<1000), bubble sort, insertion sort and selection sort have a better performance than quick sort and merge sort.
3. When the array size is large (>=5000), merge sort, quick sort extra and quick sort in place have a much better performance than bubble sort, insertion sort and selection sort.
4. Compared with other sort algorithms, std::sort performances better with almost all kinds of array size except size under 100.

2.2 Test Multiple Times With Small Array Size

For small number of input size (n < 50), all seven sorting algorithms are run many times (100 times in this case) in a single execution and compute the average time. The array size is 25 in this case. The result is listed in the table below (Table 2).

Array Size	**Bubble Sort**	**Insertion Sort**	**Selection Sort**	**Merge Sort**	**Quick Sort Extra**	**Quick Sort In-place**	**std::sort**
25	5.548	2.165	4.039	9.975	6.211	3.204	3.468

As a result. we can see that bubble sort has a stable but poor performance compared with insertion sort and selection sort, while the time for merge sort and quick sort extra almost is almost twice of that of insertion and selection sort. Surprisingly, std::sort also does a good performance when sorting small size of input. According to cppreference.com, std::sort actually use insertion sort when the input size is small. This explains why std::sort can achieve good performance with both large and small size of input because std::sort will switch the type of sorting algorithm according to the input size. For quick sort, for small input size, it will have to do partitions which requires to call of sub function in the main function. This will cost a lot unnecessary time compared with bubble sort, insertion sort and selection sort which only need to traversal the array with small input.

3. Conclusion

In conclusion, we can see how different sorting algorithms' performance is related with input size, while std::sort has a good and stable performance despite of input size.

In practice, we should pay attention to the input size, in-place and stability requirement and choose the proper sorting algorithm to solve the problem.