Ve280 Project 1 Report

Gengchen Yang

518370910088

Results

After simulation, we obtained the run time in the following table:

Table 1. Operation time v.s. Size

size	Insertion	Selection	bubble	merge	quick_extra	quick_in_place	sort
50	1.47E-05	1.85E-05	2.31E-05	6.92E-05	1.06E-04	1.14E-05	1.33E-05
500	5.85E-04	6.45E-04	1.83E-03	6.95E-04	1.28E-03	1.02E-04	1.15E-04
5000	5.74E-02	5.91E-02	1.88E-01	8.78E-03	1.51E-02	1.27E-03	1.36E-03
20000	9.07E-01	9.38E-01	3.03	3.63E-02	6.69E-02	6.01E-03	6.17E-03
50000	5.64E+00	5.83E+00	18.8	9.49E-02	1.77E-01	1.64E-02	1.66E-02
200000	8.91E+01	9.37E+01	NA	4.12E-01	7.26E-01	7.31E-02	7.59E-02

To be clearer, we plotted it as a graph, it looks like follows:

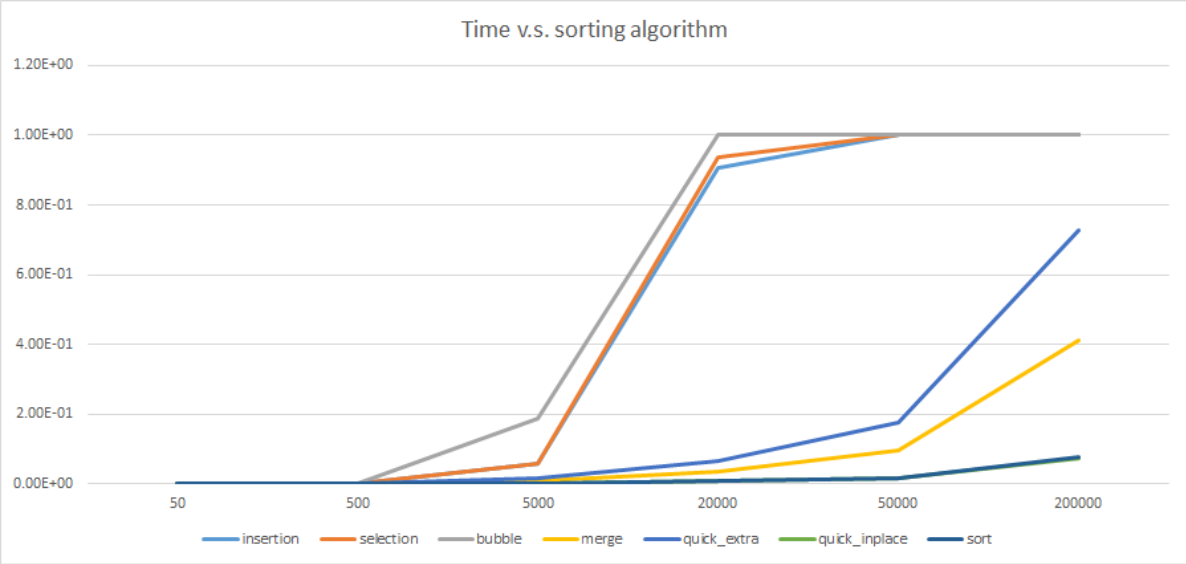

Figure 1. Operation time v.s. Size

(Note: we set 1s as a ceiling and ignore the values above it)

Analysis

We can find:

1. The operation time: Bubble > Selection > Insertion > Quick_extra > Merge > Quick_in_place = sort in most cases, when the size is very small, Merge Sort and Quick Sort (extra) takes longer than the others.
2. The increase of operation time, the first three sorting algorithms are significantly larger than others.
3. Quick sort (in place) are fast both in small sizes and large sizes.

We believe the reason that causes the above conclusions are:

1. The first three algorithms are of O(n^2) time complexity while the last four are O(nlogn), hence the increasing speed of the first three is significantly larger than the last four
2. Due to large constant multipliers, although the last four algorithms' time complexity is smaller than the first three, in small sizes, they might take longer.
3. However, std::sort is implemented with quick_in_place when the size is large and insertion when the size is small. Hence it can be fast both in large sizes and small sizes.