Project 1 Report

Gu Yucheng

November 1, 2020

Data Structures and Algorithms
VE281

Student ID 518370910168
E-mail Address *enoch2090@sjtu.edu.cn*

University of Michigan-Shanghai Jiao Tong University Joint Institute(UM-SJTU JI)

I Runtime of different Sorting Algorithms

The 6 sorting algorithms are tested along with std::sort, and the results are shown as table 1. Note for the following points:

1. For each run, a random input vector of selected length is generated and tested upon the 7 algorithms.
2. For input length less than 10,000, the listed time in the table is the average time over 5 different runs. For other inputs, the listed time is the time of one run over a random input vector.
3. For the inplace quicksort, my algorithm on JOJ has a typo at line 182, which writes "int front = left;" while the correct version is "int front = left + 1;". This will cause the pivot be swapped inside the array, and though the results are usually correct, the runtime is roughly $O\left(n^2\right)$. The data listed in the table uses the corrected version of algorithm.

The following data is tested on Ubuntu 18.04 VM inside macOS Mojave 10.14, with:

CPU : 2.9 GHz Intel Core i9

Memory : 16 GB 2400 MHz DDR4

And all time data is retrieved via C++11 chrono library.

Input	Bubble	Selection	Insertion	Merge	QSort Extra	QSort Inplace	std::sort
10	0.000001	0.0000006	0.0000002	0.00000275	0.0000072	0.0000052	0.000001
50	0.0000196	0.000012	0.0000082	0.0000178	0.0000468	0.0000372	0.0000076
1000	0.006826	0.00311	0.0020886	0.0004646	0.0009928	0.000769	0.0002254
5000	0.14976	0.0662254	0.0447926	0.0043702	0.0045666	0.003372	0.0011384
10000	0.750991	0.332596	0.205324	0.016368	0.012091	0.009207	0.003187
50000	17.638818	7.724575	5.44553	0.364424	0.056839	0.046142	0.018546
100000	71.274057	31.117099	20.566528	1.641047	0.132166	0.098491	0.038946

Table 1: Runtime of different algorithms over different size of input

Plotting the runtime versus input size yields:

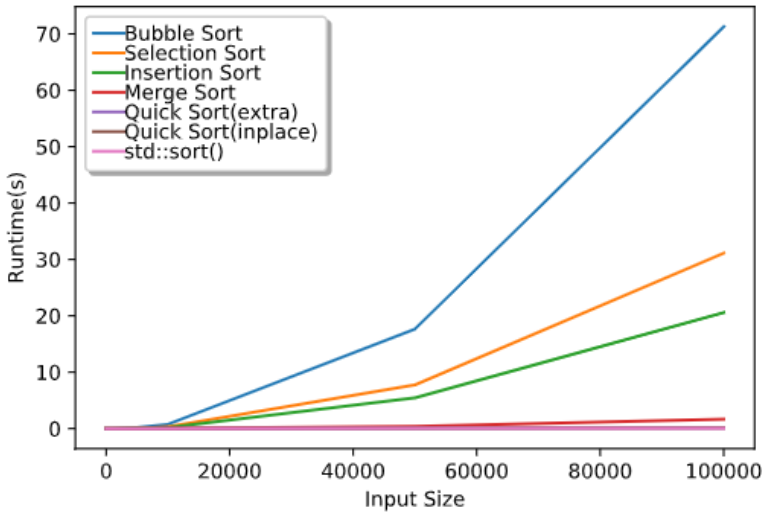

Figure 1: Runtime of different algorithms versus input size

Taking logarithm of both axis:

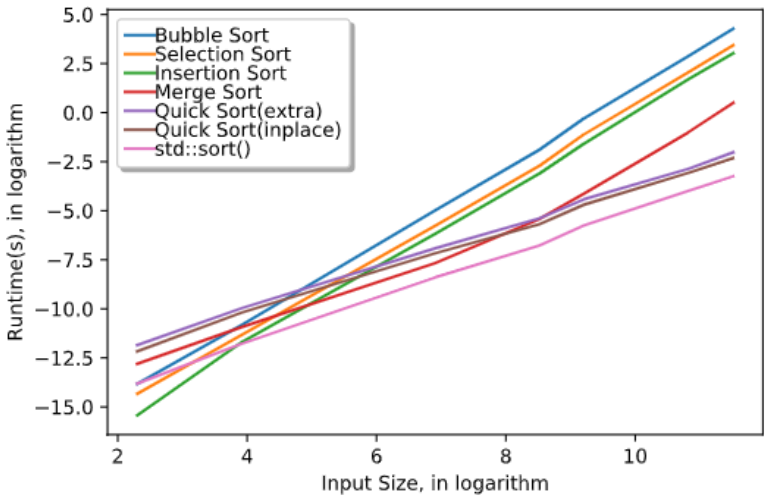

Figure 2: Runtime of different algorithms versus input size, both axis taken logarithm

II Analysis of the result

From the result, we can readoff the following conclusions:

1. For small number of input(< 50), bubble sort, selection sort and insertion sort is faster. This is because the constant factor of the

2. For large number of input(≥ 1000), merge sort and quick sort is faster. This is because they are in $O(nlogn)$ time, which is faster than $O(n^2)$ time of the first three algorithms as n grows large.

3. For std::sort, when input size is small it's close to the speed of bubble sort, selection sort and insertion sort; while when input size is large it's close to the speed of merge sort and quick sort. According to the internet, the implementation of std::sort uses a method called "Introspective Sorting". The original artical published by David Musser can be found at `http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.14.5196`. When the input is large, the algorithm uses sorting algorithm of $O(nlogn)$. When the segment to sort in some recursion is smaller than a threshold, it changes to insertion sort to achieve nearly $O(n)$ speed because the segment is probably already ordered. When the recursion level is too deep, it turns to heap sort which is also $O(nlogn)$, but has a large constant. In all, this algorithm balances the trade-offs of time and space.

Reference

1. David Musser, Introspective Sorting and Selection Algorithms, `http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.14.5196`