CS201 HW5 Sorting Algorithms Time Comparison

By Kerem Gure - S015664

1. Experimental Setup

In the experiment, in order to be consistent with the results especially when the sample size is small, for each of the sample sizes and the algorithms that were tested, 10 randomly generated samples were used (the samples were generated using TestInputGenerator.cpp.) and the average of each sample size for each algorithm type was taken into account for comparing the timing of the algorithms.

The sizes used are as follows:

- 1000
- 10000
- 100000
- 500000
- 1M
- 5M
- 10M

Also the algorithms used can be summarized as:

- SortAll:

  Uses insertion sort to sort all of the given input to the algorithm. Algorithm time O(N^2)

- SortK:

  Uses insertion sort to sort up to the index “k” then checks if next input is smaller than “k” or not then operates accordingly. Algorithm Time O(N^2)

- SortHeap

  Uses minheap structure to speed up the sorting massively. Algorithm Time O(NlogN)

- SortQuick

  Uses QuickSort and QuickSelect to sort and pick “k”th element. Algorithm Time O(N)

2. Results

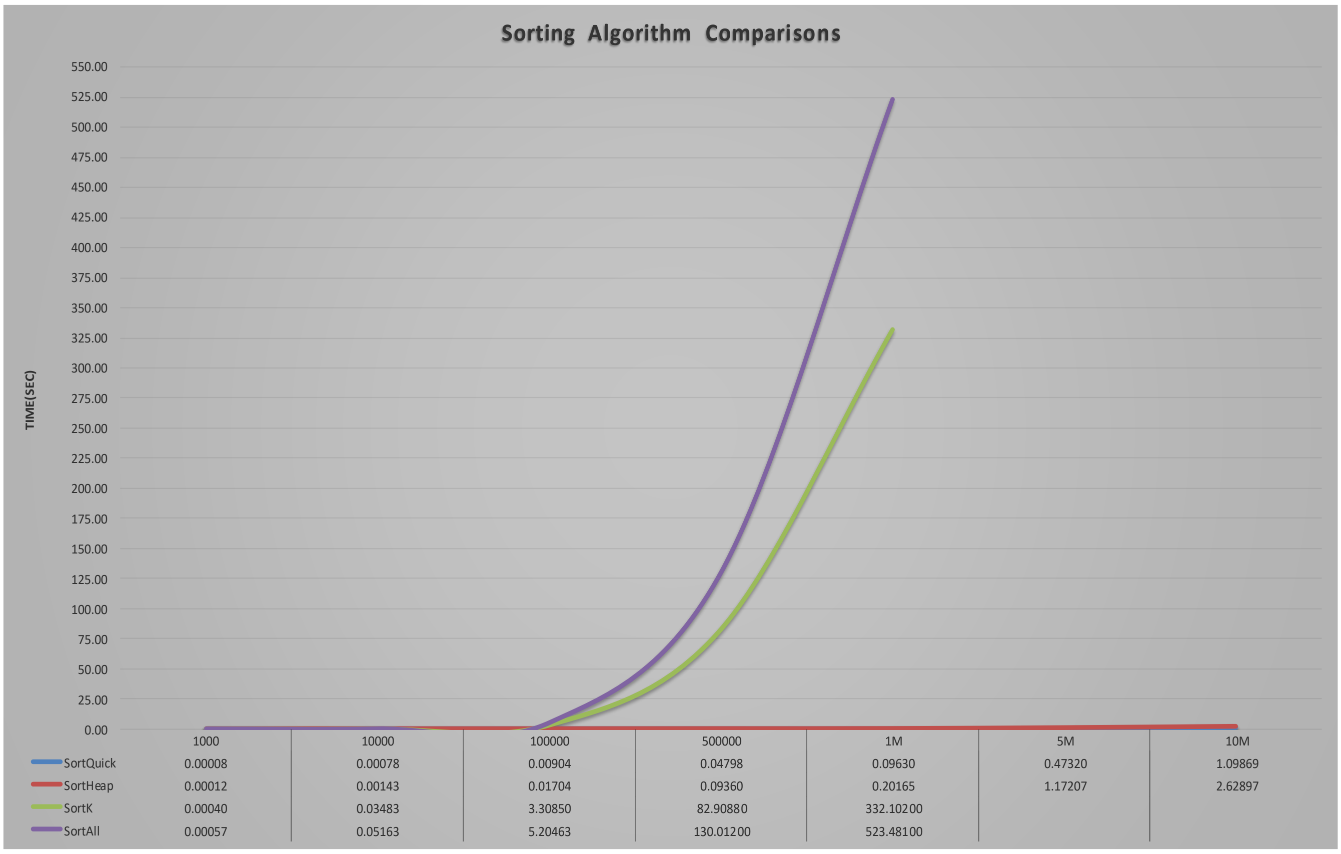
After generating and running a total of 280 test cases the following results are observed.

Samples\Algorithms	SortAll	SortK	SortHeap	SortQuick
1000	0.000570	0.000400	0.000123	0.000079
10000	0.051633	0.034830	0.001434	0.000783
100000	5.204630	3.308500	0.017044	0.009040
500000	130.012000	82.908800	0.0935970	0.047978
1M	523.481000	332.102000	0.201648	0.096302
5M	DNC*	DNC*	1.172070	0.473204
10M	DNC*	DNC*	2.62897	1.09869

Please note that all of the results are in seconds.

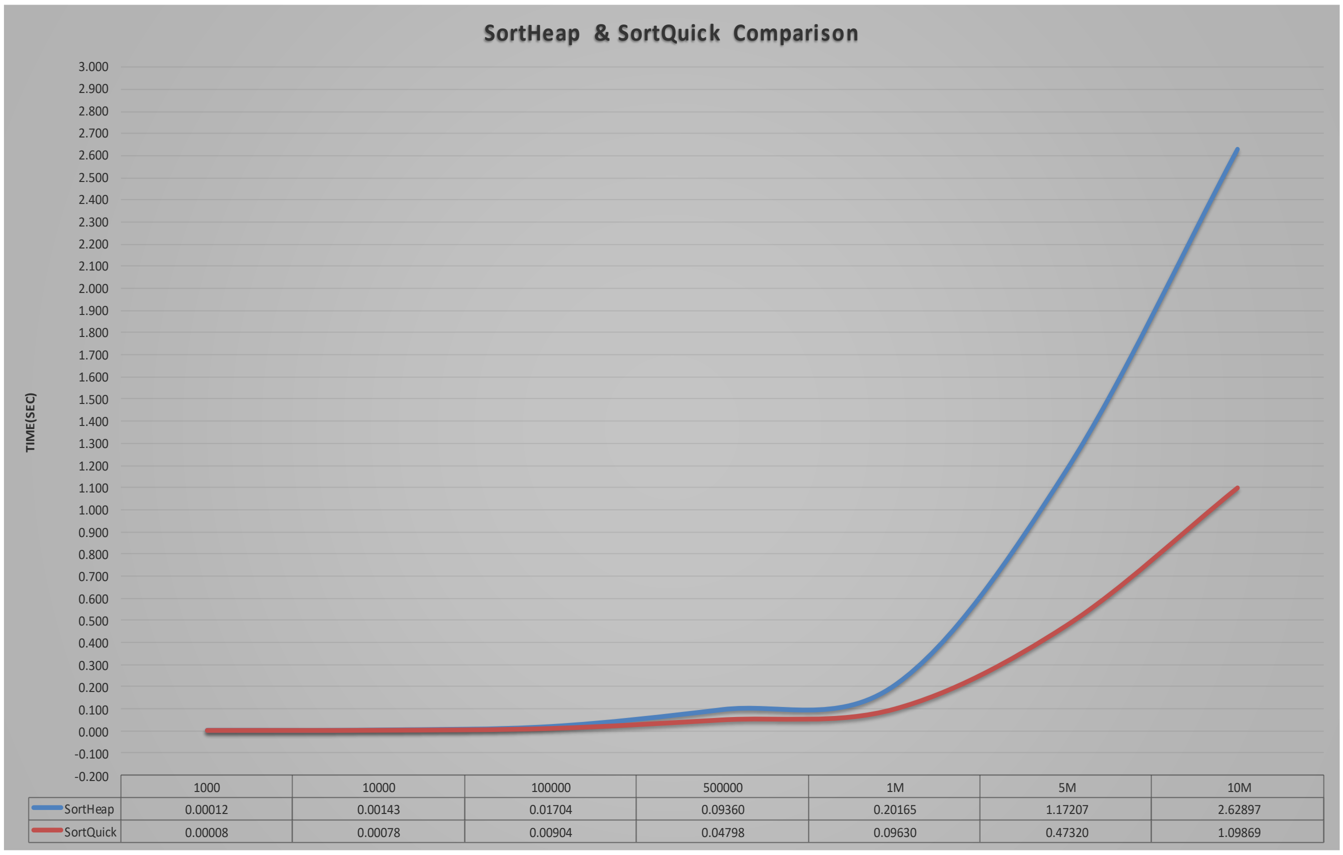
DNC*: Test did not complete under 600 seconds and therefore terminated.

In the following graph comparison of the algorithms can be seen better,



	1000	10000	100000	500000	1M	5M	10M
SortQuick	0.00008	0.00078	0.00904	0.04798	0.09630	0.47320	1.09869
SortHeap	0.00012	0.00143	0.01704	0.09360	0.20165	1.17207	2.62897
SortK	0.00040	0.03483	3.30850	82.90880	332.10200		
SortAll	0.00057	0.05163	5.20463	130.01200	523.48100		

Of course, SortHeap and SortQuick are almost indistinguishable in the graph. The following graph compares only SortHeap & SortQuick therefore will be a lot clearer to see the difference.



	1000	10000	100000	500000	1M	5M	10M
SortHeap	0.00012	0.00143	0.01704	0.09360	0.20165	1.17207	2.62897
SortQuick	0.00008	0.00078	0.00904	0.04798	0.09630	0.47320	1.09869

3. Discussion

First of all, as seen in 2nd part, SortAll and SortK runs with average of O(N^2) however, one may have noticed that SortK runs faster than SortAll, which is correct!. The reason is even though SortK uses the same sorting algorithm as SortAll which increases exponentially as seen at the first graph in the 2nd part of the report, it ignores unnecessary inputs to lower the cost of operation, one may call it an optimized version of SortAll. In addition to that, some may have also noticed that in the 2nd part of the report, the table included some DNC values, which stands for “Did not Complete” meaning that experiment were terminated after 600 seconds of running which was of course expected if one were to look at the table and pick a DNC value then pick a known value for the algorithm that DNC value belongs to, the expected result of DNC value can be calculated as follows:

Expected running time($5M_1$ | $SortAll_2$) = running time($1M_3$ | $SortAll_2$) * proportion (wanted_sample$_1$,known_sample$_3$)^2

= 523.481 * 25

= 13087.025 seconds! = 218.11 minutes!! = 3.63 HOURS!!!

1: sample size of the DNC value, 2: algorithm type for which the DNC value belongs to, 3: a sample with a known running time for the same algorithm type with DNC value.

Note that, multiplying with the proportion is special for SortAll and SortK only because of the exponentiality of their average running time. Other algorithms may require different equations for calculation.

Imagine how long would it take for 10M case…(for the curious ones its 14.54 Hours!).

Second of all, again as seen in 2nd part, SortHeap and SortQuick runs with a speed that their difference is ignorable even with huge sample sizes (such as 5M and 10M), this behavior can be explained by looking at the average running times and observing that as the input size is linearly increased so as the running time of the algorithms. SortHeap runs with an average of O(NlogN) on the other hand, SortQuick runs with an average of O(N) this means if the sample size grows 4 times bigger the proportional difference between running times will be 2, one may argue that it is a lot however, since the running the of SortQuick is very small even with huge sample sizes, logN multiples (N being the sample size such as 1000,100000…) of SortQuick, which gives the running time of SortHeap, will still be in the acceptable range for the running time of the operation. Another observation would be, average running time does not reflect fully for this experiment. That is, For example, In 10M case log10M is approximately 23 but the proportional difference for 10M case for SortHeap and SortQuick is approximately 2,40. Therefore, this difference in running times is ignorable.

Although the difference is ignorable, it does not change the fact that there is a difference! And this difference can be clearly seen at the second graph in 2nd part of the report. The graph shows the linear increase in the sample size and the linear increase of running time of the algorithms.

4. FAQ

All the tests were run on a laptop with i5-8350H CPU with Ubuntu 18.10 as OS.

One may wonder how 280 test cases were generated, tested and reported correctly. Well the answer is very simple. A Script did all the job of course!. A Python script (the source code of it will be available in github.com/keremgure) generated all the samples one by one ran the tests one by one and generated an excel file with a table that includes all the data for the experiment.