Hyun Woo Lee (z5061885), Benjamin Xia (z3462450)

COMP1927 LAB 05/06

Test theory

Each sorting algorithm will have characteristics which can be used to identify it. These include number of accesses, comparisons, insertions, moves, best case time, worst case time etc. relative to the size of the array to be sorted. However, since we have only been provided with the program binaries we cannot observe the actual data manipulations taking place, only the state of the data array before and after sorting. With this in mind, we have selected 4 characteristics which can be tested, these are:

- Time to sort a random array
- Time to sort an already sorted array
- Time to sort a reverse sorted array
- Stability

We have identified these 4 characteristics for each of the possible sorting algorithms in this lab, tabulated below:

Sort	Ave time	Already sorted	Reverse sorted	Stable
Oblivious Bubble	n^2	n	n^2	no
Bubble w/ Early exit	n^2	n	n^2	yes
Vanilla insertion	n^2	n	n^2	yes
Insertion with bin. Search	n^2	nlogn	nlogn	yes
Vanilla selection	n^2	n^2	n^2	no
Quadratic selection	n^1.5	n^1.5	n^1.5	yes
Merge sort	nlogn	nlogn	nlogn	yes
Vanilla Quick	nlogn	n^2	n^2	no
Quick sort median of three	nlogn	nlogn	nlogn	no
Randomized quick	nlogn	nlogn	nlogn	no
shell sort powers of two	n^2	n	n^2	no
Shell sort Sedgewick	nlogn	n	nlogn	no
Bogo	(n+1)!	(n+1)!	(n+1)!	no

Note that some of the possible sorts contain identical characteristics. Differentiating between these sorts will require more specialized tests. For efficiency, the initial stage of testing will focus on only the 4 characteristics above. The timed tests will be conducted by first generating a file f items to be sorted, then feeding the data into the sort program. The running time of the program is found using the time operation's 'user' time. Each test will be conducted 5 times and the average time recorded. The stability of the sort will be assessed by generating a set of data containing duplicate keys, the manually assessing the order of the output data.

At this stage, if a unique sort has not been identified, further tests will be used.

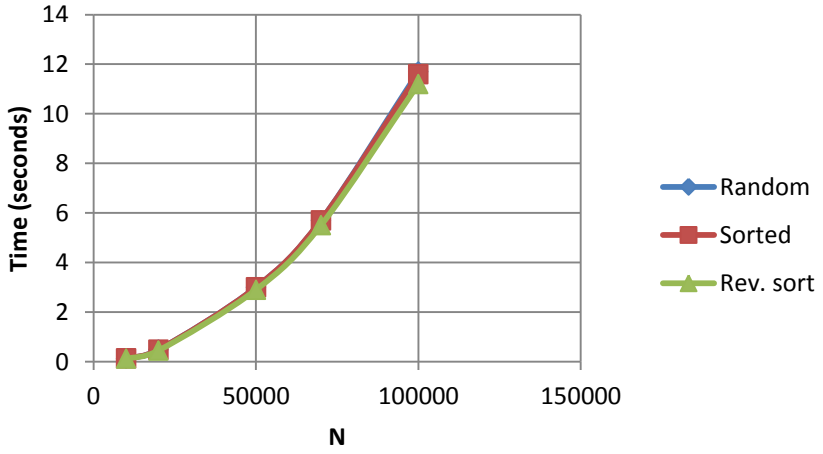
Testing Results:

Sort A

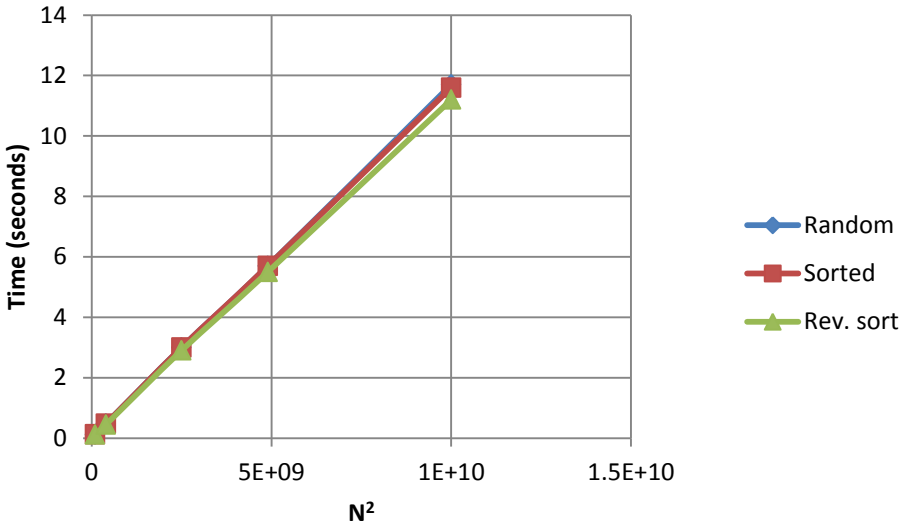
The first sort provided was run through the 4 tests and data collected. The timed sort test results are shown below:

Input size		Time to sort (seconds)		
N	**N^2**	**Random**	**Sorted**	**Rev. sort**
10000	100000000	0.12	0.13	0.12
20000	400000000	0.48	0.48	0.45
50000	2.5E+09	3	3	2.9
70000	4.9E+09	5.7	5.7	5.5
100000	1E+10	11.7	11.6	11.2

The sort was found to be unstable, as duplicate keys in the input array appeared in the output in a different order. The 3 sets of sort times are plotted against input size N, shown below:



From the above chart, the curves appear to be quadratic. To confirm this, the times were again plotted, but this time against N^2:



Since the trends are straight liens, they are indeed quadratic, implying a time complexity of $O(n^2)$ for random, already sorted and reverse sorted arrays. Combined with the unstable nature of the sort, this set of characteristics points to a VANILLA SELECTION SORT.

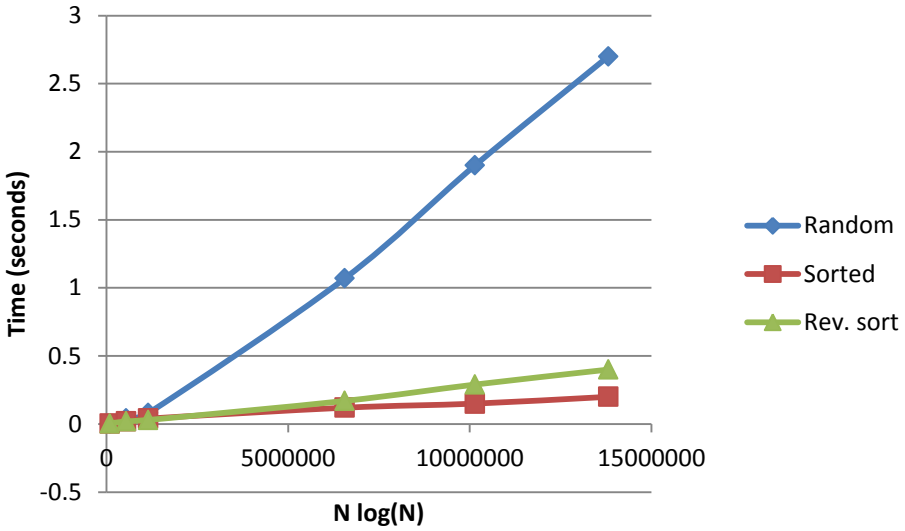
Sort B

The second sort was put through the same tests, with the data displayed below:

Input size		Time to sort (seconds)		
N	**Nlog(N)**	**Random**	**Sorted**	**Rev. sort**
10000	92103.4037	0.004	0.004	0.004
50000	540988.914	0.04	0.02	0.02
100000	1151292.55	0.08	0.04	0.03
500000	6561181.69	1.07	0.12	0.17
750000	10145871.4	1.9	0.15	0.29
1000000	13815510.6	2.7	0.2	0.4

The sort was found to be unstable using the method described previously. Compared to the sort A results, sort B was much faster, indicating that it was probably a O(nlogn) sort. The 3 sets of sort times are plotted against input size Nlog(N), shown below:

Hyun Woo Lee (z5061885), Benjamin Xia (z3462450)



From the linear trends, it can be assumed that the time complexity of sortB is nlogn for random, sorted and reverse sorted data. Based on this, as well as the fact that it is unstable, it can be concluded that sort B is a quick sort (either Median of Three or randomized). Median of three tends to be slightly faster than randomized pivot selection, however without another sort to compare against, it is impossible to differentiate the two by speed alone, as it will depend on the hardware the code is run on as well as the data provided. Writing our own median-of-3/randomized quicksort would also not be a reliable way to test, as we would not know how efficiently the provided sort performs swaps, compares, random number generation, etc compared to our own.