COMP2521 Sort Detective Lab Report

by Liam Lau (z5288234), Xerox Chan (z5289835)

In this lab, the aim is to measure the performance of two sorting programs, without access to the code, and determine which sorting algorithm each program uses.

Experimental Design

1. Firstly, we should test the stability of the sort algorithms. There should be 2 items that are different but equal under comparison, such as same items with different ids. If the ids/keys remain in original order then the algorithm is stable, which may be implying bubble sort etc..

2. Manual testing for whether the sorting method stored original order of values. With

randomly generated data, and manually changing the values (changing to same value

to be sorted to see order), we tested using sortA and sortB Any change in the order

of the letters indicate that the sorting methods change the order of list somehow

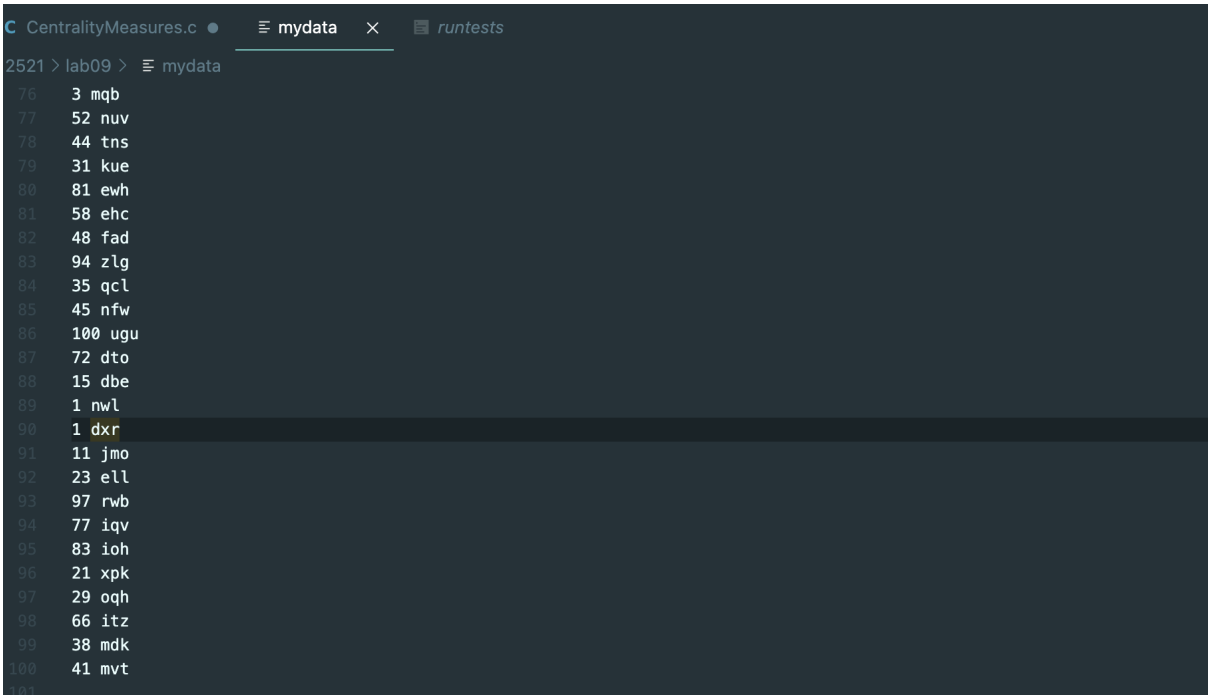
before sorting

Time complexity

Testing 20000, 40000, 60000, 80000, 100000 elements three times each with ascending, descending and in random generated lists to see how long sortA and sortB takes on average.

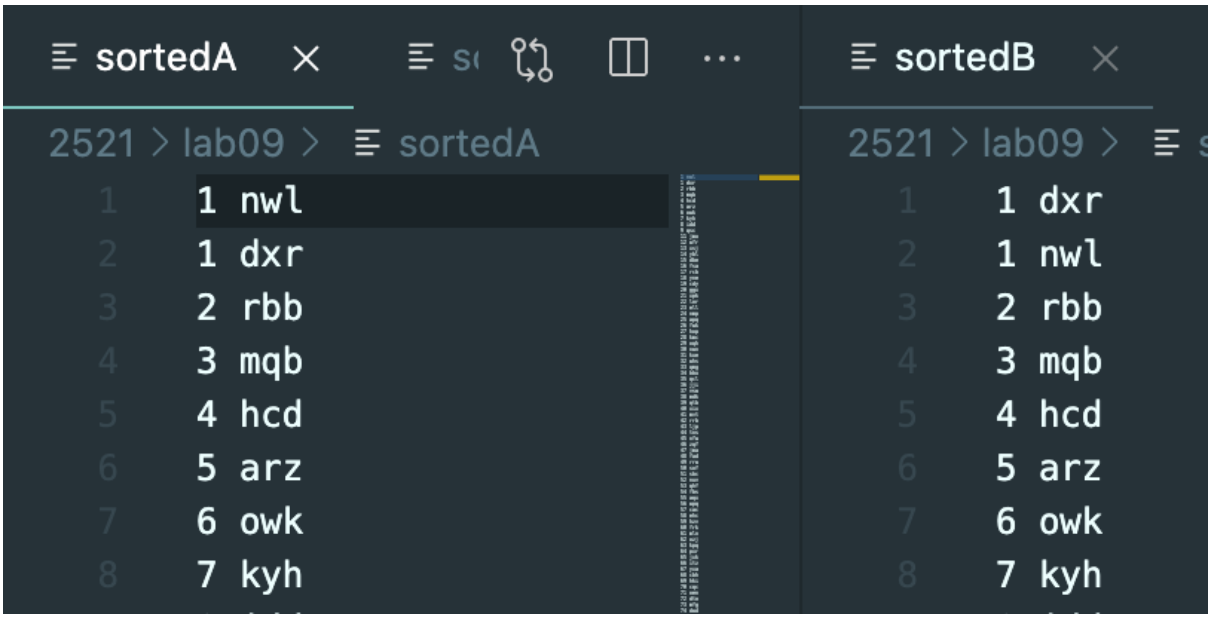
Experimental Results

Stability test



notice the order of keys for 1

results:



For Program A, we observed that ...

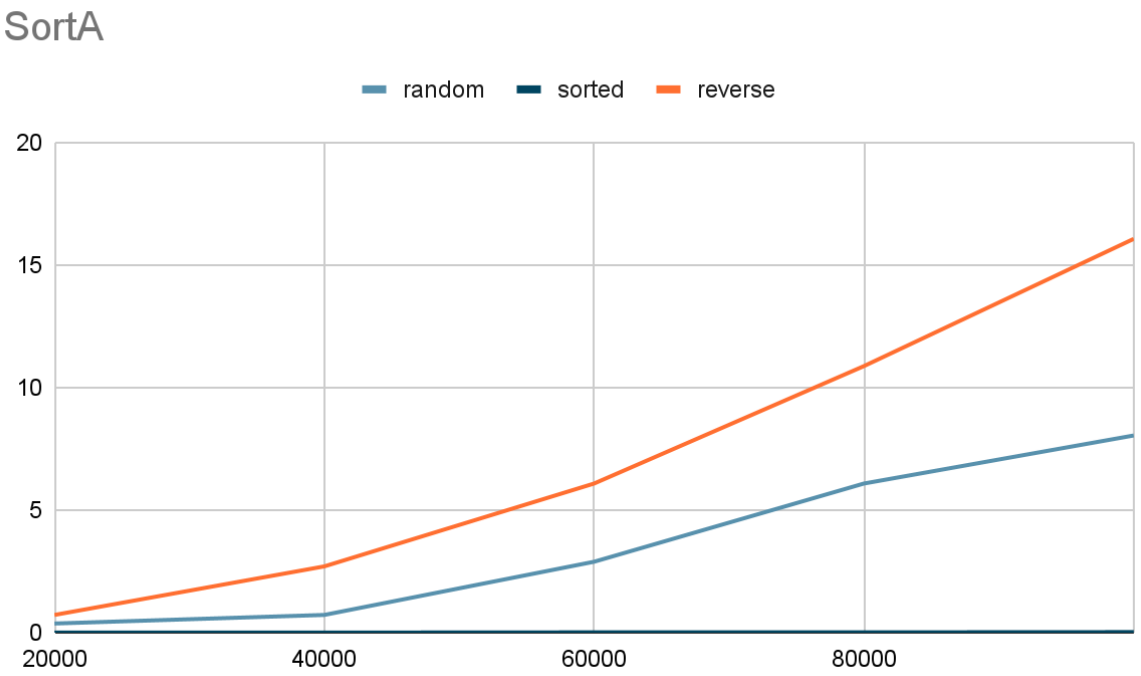
Stable as the keys remain in order, so it must be insertion, merge, bubble or randomized quick sort.

For Program B, we observe that…

Unstable as the keys are not in order and have been shuffled

Time complexity result

SortA			
Input size	Initial order	average time(seconds)	Number of runs
20000	random	0.37	3
20000	sorted	0.00	3
20000	reverse	0.72	3
40000	random	1.32	3
40000	sorted	0.00	3
40000	reverse	2.70	3
60000	random	2.89	3
60000	sorted	0.01	3
60000	reverse	6.08	3
80000	random	6.08	3
80000	sorted	0.01	3
80000	reverse	10.87	3
100000	random	8.04	3
100000	sorted	0.02	3
100000	reverse	16.06	3



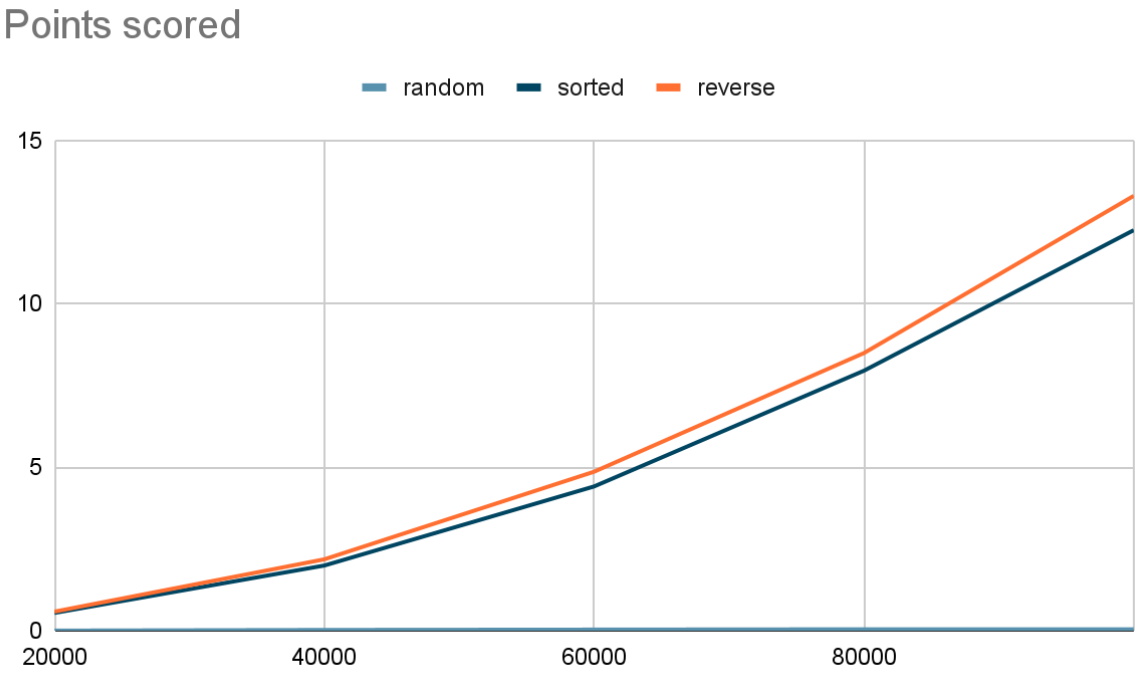
From the table above, we can see that sortA time increases with the size of input. The sorted input always uses the shortest time and the reverse input takes the longest time to sort.

Since the time used for random, sorted and reverse have an obvious difference, it cannot be merge sort or randomized quick sort as time complexity for merge sort is always O(n*log(n)). That means the time taken would be similar.

Since there is a considerable difference between random order and reverse order, it is unlikely that the algorithm is bubble sort as bubble sort should have similar performance with reverse and random order.

Therefore, there is a high chance that sortA is insertion sort.

SortB			
Input size	Initial order	average time(seconds)	Number of runs
20000	random	0.00	3
20000	sorted	0.54	3
20000	reverse	0.58	3
40000	random	0.02	3
40000	sorted	1.99	3
40000	reverse	2.18	3
60000	random	0.03	3
60000	sorted	4.41	3
60000	reverse	4.86	3
80000	random	0.04	3
80000	sorted	7.95	3
80000	reverse	8.49	3
100000	random	0.04	3
100000	sorted	12.25	3
100000	reverse	13.30	3



From the above table, we can see that sortB time increases with the size of input. We can also see that sortB works best when the input is random and it works badly when the input is reverse or sorted. This property of the algorithm and using the fact that sortB is unstable. There is a high chance that sortB is naive quicksort.

Conclusions

On the basis of our experiments and our analysis above, we believe that

- sortA implements the *insertion* sorting algorithm
- sortB implements the *naive quicksort algorithm*