# COMP2521 Sort Detective Lab Report

**by Weihan Wang(z5250416) and Zeal Liang(z5325156)**

In this lab, we are going to observe the behavior of the two programs when dealing with different sequences. The aim is to determine which sorting algorithm each program use.

## Experimental Design

We measured the execution time of the two programs dealing with sequences in different sizes and in different order such as sorted, reversed and random. Sorting stability is also recorded for further discussion.

We use these testing cases because different sorting algorithms differ significantly when the input size is large enough such as O(n^2).
By using different kinds of sorted sequences and the stability, we can figure out which quick sort and which sort algorithm does the program use.

Regarding of the timing uncertainty in Linux system, we repeat the same case 10 times and calculate the average time for each case.

## Experimental Results

For program A, the stability tests indicate that the positions of the sequence with the same key do not change after a specific algorithm. From the chart below, we discover that the time used for random and reversed is nearly the same when the input size is relatively small. But the time for reversed one becomes longer as the input size increases.

However, the time cost for the sorted one is close to zero which is extremely small compared with the other two. The figure recorded indicate the algorithm is stable and the worst case appears when the sequence is random or reversed.

The average time cost: sorted<<random<reversed.

For program B, the stability tests indicate that the positions of the sequence with the same key change after a specific algorithm. From the chart below, we discover that the time used for sorted and reversed is nearly the same when the input size is relatively small. But the time for reversed one becomes longer as the input size increases.

However, the time cost for the random one is especially small compared with the other two. The figures recorded indicate the algorithm is unstable and the worst case appears when the sequence is sorted or reversed.

Our initial guess is Naive quicksort or median-of-three quicksort.But when we change the leftmost element in the data, its sorting time will not be affected.

Then we changed the first,middle and last elements and found that the sorting time is not affected either, so it should be Randomised quicksort

The average time cost: random<<sorted<reversed.

## Conclusions

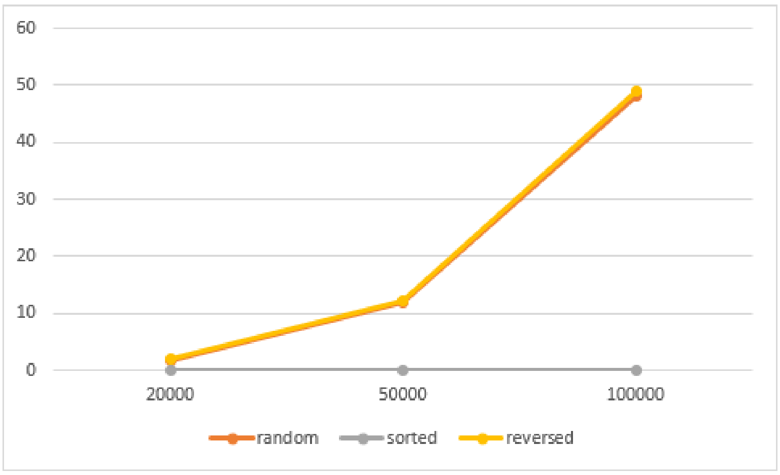
On the basis of our experiments and our analysis above, we believe that:

- sort A implements the Bubble sorting algorithm.
- sort B implements the Randomised quicksort sorting algorithm.

## Appendix:

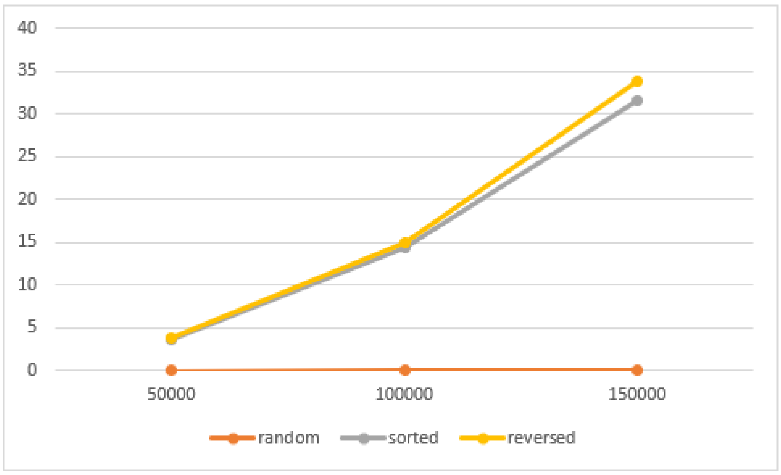
sortA

| Size | Initial order | Times | Average time | Stability |
|---|---|---|---|---|
| 20000 | random | 10 | 1.82 | yes |
| 20000 | sorted | 10 | 0.01 | yes |
| 20000 | reversed | 10 | 1.95 | yes |
| 50000 | random | 10 | 11.96 | yes |
| 50000 | sorted | 10 | 0.01 | yes |
| 50000 | reversed | 10 | 12.21 | yes |
| 100000 | random | 10 | 48.29 | yes |
| 100000 | sorted | 10 | 0.01 | yes |
| 100000 | reversed | 10 | 49.07 | yes |



sortB

| Size | Initial order | Times | Average time | Stability |
|---|---|---|---|---|
| 50000 | random | 10 | 0.02 | no |
| 50000 | sorted | 10 | 3.54 | no |
| 50000 | reversed | 10 | 3.75 | no |
| 100000 | random | 10 | 0.03 | no |
| 100000 | sorted | 10 | 14.43 | no |
| 100000 | reversed | 10 | 14.99 | no |
| 150000 | random | 10 | 0.05 | no |
| 150000 | sorted | 10 | 31.63 | no |
| 150000 | reversed | 10 | 33.72 | no |


random
sorted
reversed