COMP2521 Sort Detective Lab Report

By Tony Ding (z5218689) and Pratik Napit (z5311922)

This experiment will aim to determine which sorting algorithm has been used without access to the code. There are 8 different algorithms for sorting, and these can be classified as either stable or unstable. The report will be divided into 3 sections: the experimental design, experimental results, and the Appendix.

Phase 1: Experimental Design

Testing Stability:

The first stage of testing will need us to determine whether the sorting algorithm is a stable or unstable algorithm. This will involve testing with Sort A and Sort B with varying values in 1 key, whilst keeping the value of the other key to be the same. This will give us an indication of whether its stable or not. After testing with 1 key, an analysis of the order is to be made- if the order is preserved after sorting then we can assume it is stable, if not, it would be unstable. 3 different arrays will be made to ensure the state of its stability to ensure reliability of the experimental design.

If the sorting is stable, we can narrow the algorithms down to bubble sort, insertion sort and merge sort. If the sorting is not stable, the algorithms can either be selection sort, naïve quicksort, median-of-three quicksort, randomised quicksort or bogosort.

Volume Testing: Performance Analysis

The next stage of testing is volume testing, where we input a large list of 10 000 to 100 000 values. Each value will be incremented by 10 000. This will be done for algorithms sortA and sortB as well. The size of inputs will vary, depending on the time complexities and speeds of the algorithms. If an algorithm has a faster time complexity, it might not be accurate to test for inputs (10 000 to 100 000), and might require a greater input (e.g. from 100 000- 1 000 000). This will be determined as we actually test them, however for any case, 10 different input sizes will be tested.

To account for worst case, average case and best-case scenarios, there will be tests with values in ascending order, descending order and random order. In order to ensure scientific reliability, 5 tests will be done and an average of all of the tests for each type (ascending, descending and

random) will be taken. To help better visualise, an excel chart will then be constructed in order to analyse its O(N) time complexity. The y-axis, or the output value of the excel chart will contain the time taken in seconds, whilst the x-axis will have the size of input.

Phase 2: Experimental Results

Testing stability:

Test 1:

Original	**Sort A**	**Sort B**
5 arz	5 arz	5 rbb
5 nwl	5 nwl	5 mqb
5 hcd	5 hcd	5 hcd
5 mqb	5 mqb	5 nwl
5 owk	5 owk	5 owk
5 rbb	5 rbb	5 arz

Test 2:

Original	**Sort A**	**Sort B**
8 ssz	8 ssz	8 oef
8 ffs	8 ffs	8 ffs

8 hea	8 hea	8 ssz
8 htc	8 htc	8 htc
8 oef	8 oef	8 vdr
8 vdr	8 vdr	8 hea

Test 3:

Original	**Sort A**	**Sort B**
1 taz	1 taz	1 taz
1 hss	1 hss	1 ota
1 ota	1 ota	1 sxc
1 sxc	1 sxc	1 ubr
1 uwf	1 uwf	1 hss
1 ubr	1 ubr	1 uwf

From the 3 tests, we can say with certainty that Sort A is a stable sort as we can see that the original list order is maintained, whilst Sort B is an unstable sort as the list order is randomised, and order is not preserved.

Thus, sort A can be either a:

- Bubble sort
- Insertion Sort

· Merge Sort

Sort B can be either:

· Selection Sort

· Naïve Quick Sort

· Median of three quick sort

· Randomised quicksort

· Bogosort

Volume Testing: Performance Analysis

Sort A:

Input Size	**Sorted Order** (s)	**Reverse Order** (s)	**Randomised Order** (s)
10 000	0.15	0.17	0.17
20 000	0.61	0.56	0.61
30 000	1.32	1.26	1.33
40 000	2.32	2.36	2.30
50 000	3.624	3.47	3.61
60 000	5.21	5.03	5.22
70 000	7.14	6.82	7.10
80 000	9.23	8.89	9.32
90 000	11.65	11.62	11.71

100 000	14.37	13.97	14.41

From looking at these results, the timings increase proportionally with the increments in input size, somewhat quadratically (as it is not linearly). As input size increases, we can see that the randomised and the ascending order are nearly the exactly the same, whilst when it is descending order, it is slightly quicker than the others.

From this, we know it can't be bubble sort as if we were to compare the reverse order and sorted order, we should notice that there would be a relatively considerable difference between them. This is due to bubble sort having only to iterate once in a sorted array and would thus have a time complexity of O(n), whilst in the worst-case scenario, where it would be reversed it would have a complexity of O(n^2). In this example, we see that the reverse order is in fact, slightly shorter than the sorted order. Similarly, for a sorted list, insertion sort should be faster than the reverse order (should be considerably larger, comparing O(n) to O(n^2) respectively), since for a reverse order, each iteration would shift the entire sorted subsection of the array before inserting the next element.

Therefore, it cannot be a bubble sort or an insertion sort and must be a **merge sort**. To confirm, looking at the time for random, sorted and reverse, we can observe that they are almost the same. In addition, if we look at the excel spreadsheet, we notice that they grow relatively the same, and have relatively similar values.

Sort B

Input Size	**Sorted Order (s)**	**Reverse Order(s)**	**Randomised Order (s)**
100 000	0.02	0.02	0.03
200 000	0.04	0.04	0.07
300 000	0.06	0.06	0.1
400 000	0.07	0.08	0.15

500 000	0.10	0.11	0.18
600 000	0.12	0.13	0.20
700 000	0.14	0.16	0.28
800 000	0.16	0.17	0.30
900 000	0.18	0.19	0.33
1 000 000	0.23	0.21	0.40

We know from the stability of this algorithm (unstable) that it can be of the following algorithms:

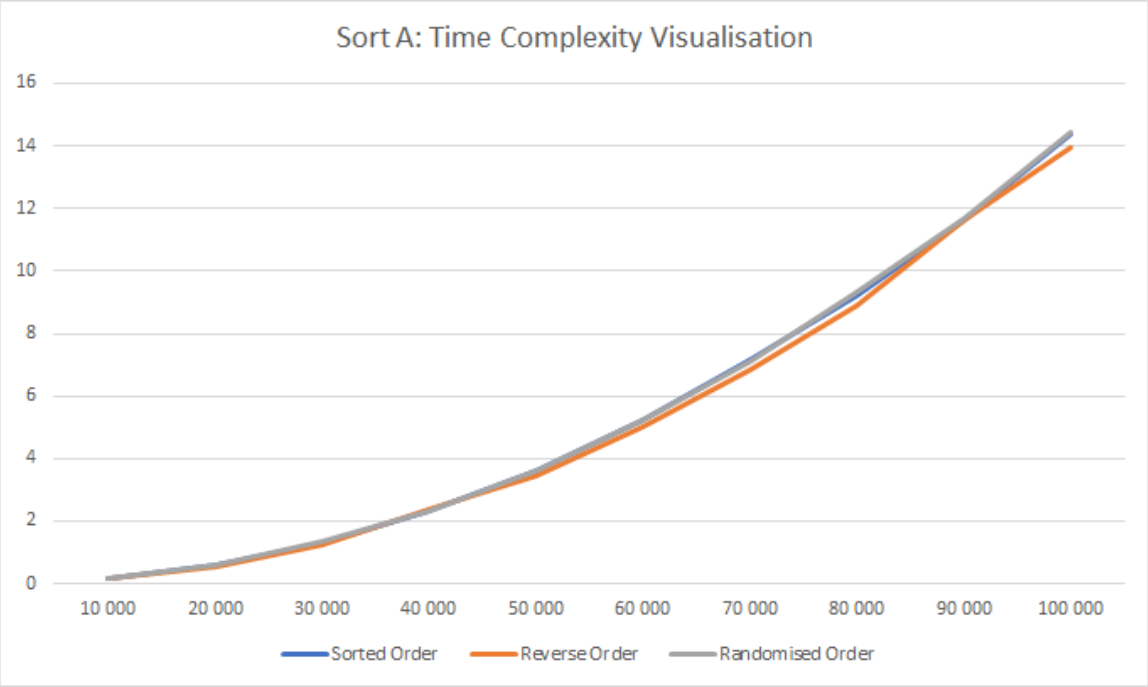
- Selection Sort
- Naïve Quick Sort
- Median of three quick sort
- Randomised quicksort
- Bogosort

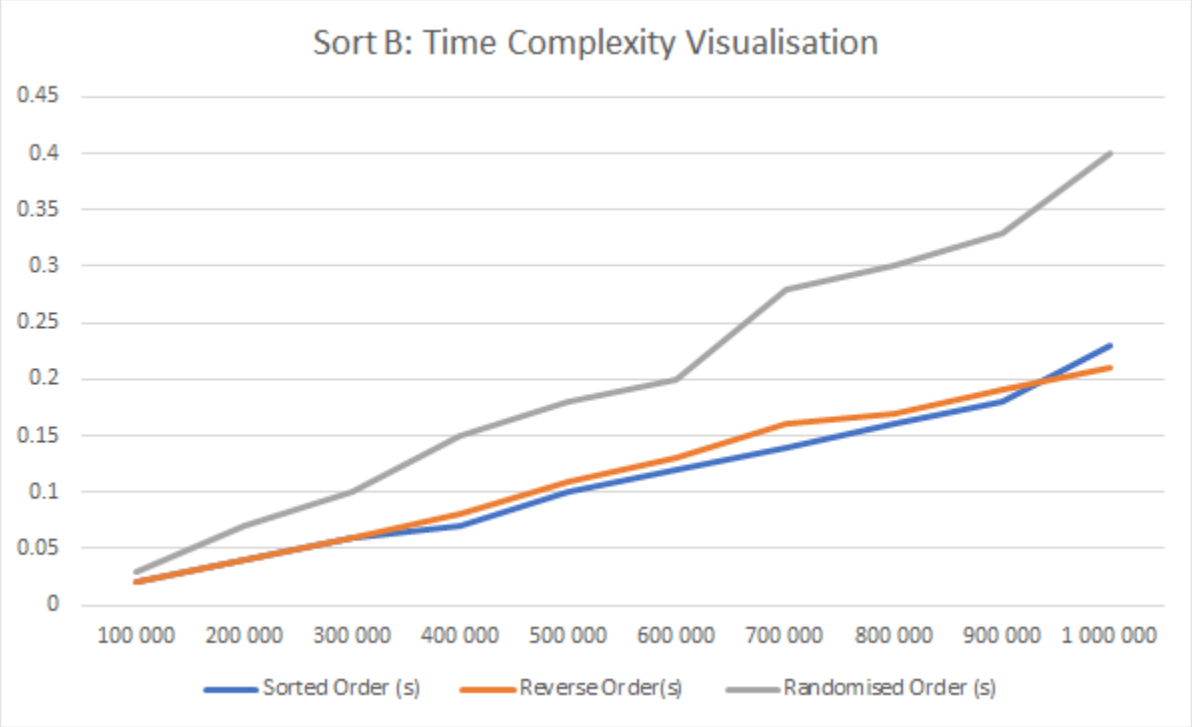
Since the time complexity grows quasi linearly with output for both sorted, reversed and random order, this algorithm is not a selection sort which has time complexity of O(n^2), no matter what. Further, it is not Bogosort because it has a time complexity of O((n+1)!) in average performance. This very high speed of the algorithms also give evidence towards a quicksort algorithm. Out of the quick sort algorithms, if Naive quicksort takes in the last element in the list as the pivot element, it will therefore have a time complexity of O(nlogn) if the array is sorted, but O(n^2) if it is in reverse sort. Thus, looking at the quasi linearity of our results for both sorted and reverse, our algorithm is unlikely to be Quicksort with a last element pivot.

Finally, between the median of 3 quicksort and randomised quicksort, since our results show that if the sort was randomised, the time complexity is visibly worse than for sorted or reverse sort arrays, we can conclude the algorithm is most likely a randomised quicksort. This is because if we were using the median of 3 quicksort, it has very similar random sort time complexity as reverse and random, this is not evident in our analysis.

Instead, quicksort with a random pivot has very similar linear time complexity for sorted and reversed sorts, and a longer time complexity for random sorts - as reflected in our analysis.

Appendix





Here, the time (y-axis) is plotted against the number of inputs (x-axis)