Sort Detective Report

By Chris Joy, Allan Chen & Alexander Nguyen

In this report we aim to deduce the sorting algorithm used by type programs, without knowing their implementation details. In order to do this, we will need to measure the running time of each program and correlate them with a time complexity.

Sorting Algorithms

The table below contains various sorting algorithms discussed during the lectures with their relevant time complexities.

Algorithm	Best	Average	Worst	Stability
Bubble Sort	$O(n^2)$	$O(n^2)$	$O(n^2)$	Yes
Bubble Sort EE	$O(n)$	$O(n^2)$	$O(n^2)$	Yes
Selection Sort	$O(n^2)$	$O(n^2)$	$O(n^2)$	No
Insertion Sort	$O(n)$	$O(n^2)$	$O(n^2)$	Yes
Shell Sort, Po4	$O(n \log(n))$	-	$O(n^2)$	No
Shell Sort, SW	$O(n \log(n))$	$O(n^{7/6})$	$O(n^{4/3})$	No
Bogo Sort	$O(n)$	$O(n * n!)$	$O(inf)$	No

Experiment Design

Our experiments will essentially consist of two phases.

1. Verify our sorting programs behave correctly (i.e. they're actually sorting items.
2. Measure runtime performance of both algorithms, with various types of inputs.

In order to conduct these experiments, we'll write and run a shell script, consisting of various tasks needed, in order to complete the requirements above.

We'll also need to consider various factors when writing and running the experiments, such as:

- Device used to run experiment (lab server / login server)
- System time vs Program time vs User time etc
- Time used to write to file
- Seed used when randomly generating a list of numbers

During testing, the MODE (most common item) will be added to the final results table.

Correctness Analysis

In order to check if the given sorting programs are correct, we compare the output of each program with that of the output produced by the UNIX sort program. We used the diff program to compare these outputs. I.e.

```
diff <(./gen $size $flag | tail -n -$((size-1)) | ./$program) <(./gen $size $flag | tail -n -$((size-1)) | sort -n)
```

This command would produce nothing if both outputs after sorting are exactly the same. We tested the outputs of both sorting programs using all flags available (i.e. A, R, & D) and on different batch sizes (100, 1000, 10000, 100000).

Performance Analysis

In our performance analysis, we measured how each program's execution time varied. We used the following kinds of inputs to test the following cases:

- FLAG=A (Ascending order) -> Best case
- FLAG=R (Random order) -> Average case
- FLAG=D (Descending order) -> Worst case

We used these tests due to the following assumptions:

- Most sorting algorithms use the least amount of comparisons and swaps when items are already sorted.
- Conversely, sorting algorithms take the most amount of time when items are in descending order.

Because of the way timing works on Unix/Linux, it was necessary to repeat the same test multiple times. After running the tests multiple times (5 times for each test), we took the **mode** (most common score) as the final result.

We were able to use up to quite large test cases without storage overhead because

a. We had a data generator that could generate consistent inputs to be used for multiple test runs.
b. We had already demonstrated that the program worked correctly, so that there was no need to check the output.

Sort A Timing Data

Number of Items	Best-case time (s)	Average-case time (s)	Worst-case time (s)
100	0.00	0.00	0.00
1000	0.00	0.00	0.00
10000	0.00	0.08	0.14
20000	0.00	0.31	0.63
40000	0.00	1.25	2.36
80000	0.02	4.57	9.25
160000	0.04	18.15	37.00
200000	0.06	27.56	57.62

Sort B Timing Data

Number of Items	Best-case time (s)	Average-case time (s)	Worst-case time (s)
100	0.00	0.00	0.00
1000	0.00	0.00	0.00
10000	0.10	0.11	0.12
20000	0.51	0.44	0.44
40000	1.85	1.78	1.80
80000	7.02	7.23	7.24
160000	28.64	28.00	29.22
200000	42.92	43.50	44.65

Stability Analysis

We also tested the stability of both sorting algorithms by exploiting the fact that both programs only sort items using the first character of each line. When generating out testing dataset, we appended letters ranging from A to E, in order to show the original order, they were added. We ran tests with batches of 100 and 1000 items, but due to the limited space available on this document, we'll show the first 150 items.

Test Dataset (Unorder Prefix, Ordered Suffix)

1a 2a 3a 4a 5a 6a 7a 8a 9a 10a 11a 12a 13a 14a 15a 16a 17a 18a 19a 20a 21a 22a 23a 24a 25a 26a 27a 28a 29a 30a 31a 32a 33a 34a 35a 36a 37a 38a 39a 40a 41a 42a 43a 44a 45a 46a 47a 48a 49a 50a 51a 52a 53a 54a 55a 56a 57a 58a 59a 60a 61a 62a 63a 64a 65a 66a 67a 68a 69a 70a 71a 72a 73a 74a 75a 76a 77a 78a 79a 80a 81a 82a 83a 84a 85a 86a 87a 88a 89a 90a 91a 92a 93a 94a 95a 96a 97a 98a 99a 100a 1b 2b 3b 4b 5b 6b 7b 8b 9b 10b 11b 12b 13b 14b 15b 16b 17b 18b 19b 20b 21b 22b 23b 24b 25b 26b 27b 28b 29b

SortA Ordered Results

1a 1b 1c 1d 1e 2a 2b 2c 2d 2e 3a 3b 3c 3d 3e 4a 4b 4c 4d 4e 5a 5b 5c 5d 5e 6a 6b 6c 6d 6e 7a 7b 7c 7d 7e 8a 8b 8c 8d 8e 9a 9b 9c 9d 9e 10a 10b 10c 10d 10e 11a 11b 11c 11d 11e 12a 12b 12c 12d 12e 13a 13b 13c 13d 13e 14a 14b 14c 14d 14e 15a 15b 15c 15d 15e 16a 16b 16c 16d 16e 17a 17b 17c 17d 17e 18a 18b 18c 18d 18e 19a 19b 19c 19d 19e 20a 20b 20c 20d 20e 21a 21b 21c 21d 21e 22a 22b 22c 22d 22e 23a 23b 23c 23d 23e 24a 24b 24c 24d 24e 25a 25b 25c 25d 25e 26a 26b 26c 26d 26e 27a 27b 27c 27d 27e 28a 28b 28c 28d 28e

SortB Ordered Results

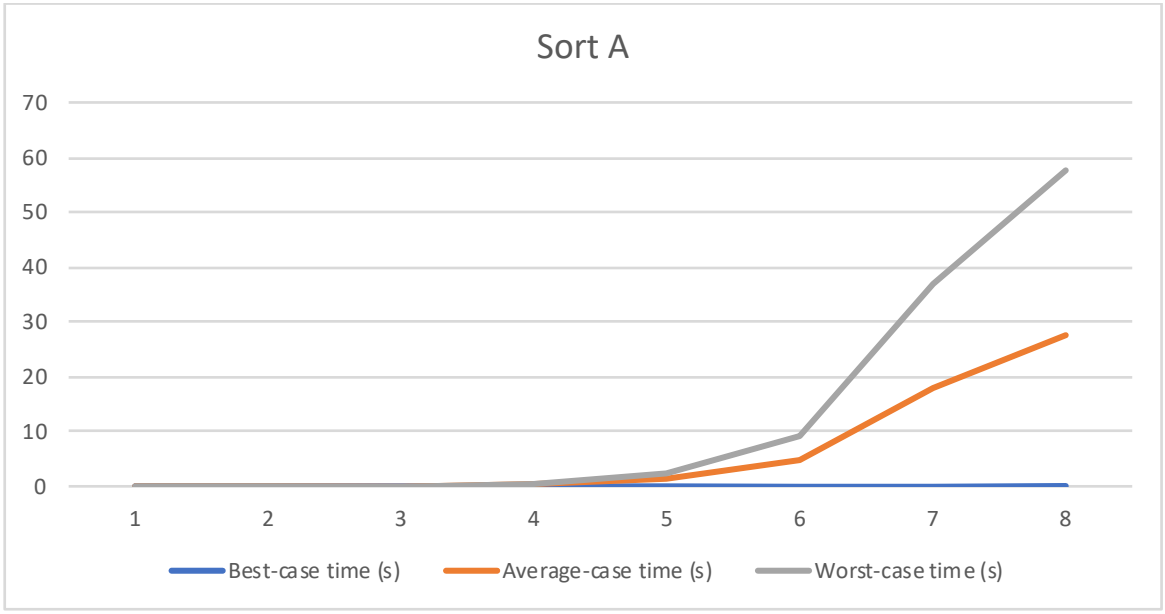
1a 1b 1c 1d 1e 2a 2b 2c 2d 2e 3b 3a 3c 3d 3e 4b 4c 4a 4d 4e 5b 5c 5d 5a 5e 6a 6b 6c 6d 6e 7a 7b 7c 7d 7e 8b 8a 8c 8d 8e 9b 9c 9a 9d 9e 10b 10c 10d 10a 10e 11a 11b 11c 11d 11e 12b 12a 12c 12d 12e 13b 13a 13c 13d 13e 14b 14c 14a 14d 14e 15b 15c 15d 15a 15e 16a 16b 16c 16d 16e 17b 17a 17c 17d 17e 18b 18c 18a 18d 18e 19b 19c 19a 19d 19e 20b 20c 20d 20a 20e 21a 21b 21c 21d 21e 22b 22a 22c 22d 22e 23b 23c 23a 23d 23e 24b 24c 24d 24a 24e 25b 25c 25d 25a 25e 26b 26c 26d 26a 26e 27a 27b 27c 27d 27e 28a 28c 28b 28d

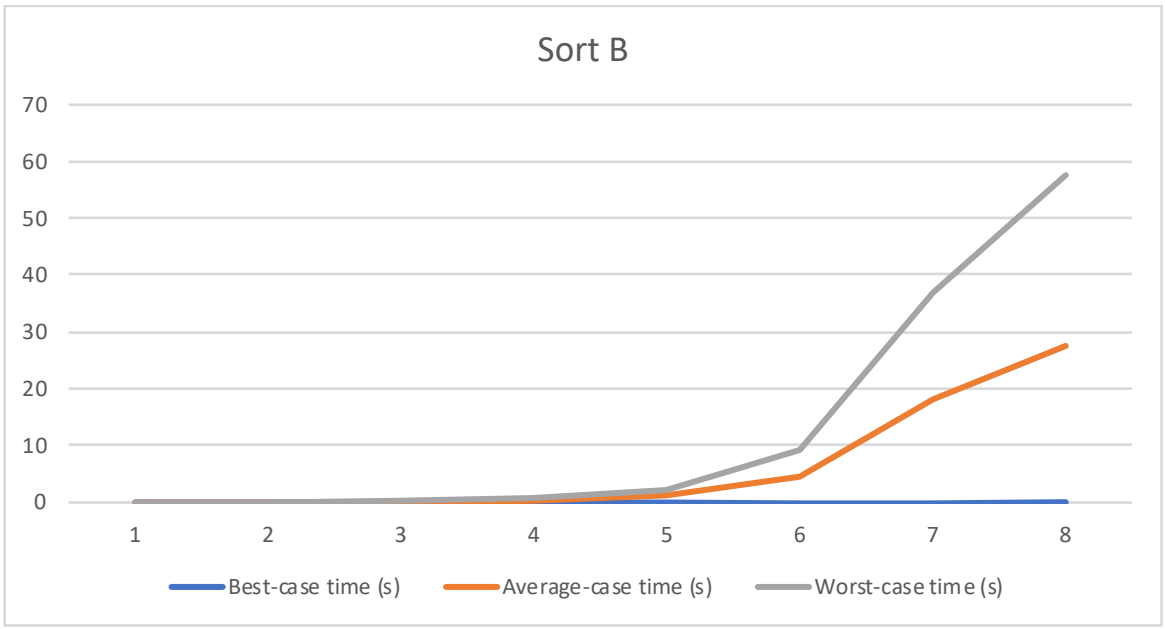
Experimental Results

Correctness Experiments

There was no difference between the output of the given programs and outputs produced by the UNIX sort program, hence they're correct.

Performance Experiments





Key
Blue - Best
Yellow – Worst Case

Stability Experiments

With SortA, we saw that the key ordering (suffix values in this case), had been maintained (i.e. A because B, B before C …). However, with SortB, we saw that the key order hadn't been maintained on some instances. For example, if we look at the keys with the prefix 25:

- 25b 25c 25d 25a 25e, …

We see that list above is sorted on the prefix, but the suffix ordering has been changed.

Conclusions

Correctness

- Sort A – Yes
- Sort B – Yes

Stability

- Sort A – Stable
- Sort B – Unstable

Algorithms

- Sort A – Bubble Sort EE OR Insertion Sort
  - Best case: $O(n)$
  - Worst case: $O(n)$
- Sort B – Selection Sort
  - Best case: $O(n^2)$
  - Worst case: $O(n^2)$