— COMP2521 19T0 —

Sort Detective Report

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

Experimental Design

There are two aspects to our analysis:

- determine that the sort programs are actually correct, and
- measure their performance over a range of inputs.

Correctness Analysis

To determine correctness, we tested each program on the following kinds of input:

- Sorting ascending list of numbers
- Sorting descending list of numbers
- Sorting random list of numbers

We chose these inputs because:

- Ascending order, will enable use to determine how the sorting algorithms behave when sorting data that is already sorted. We know that insertion sort/optimised bubble has a time complexity of O(n) when sorting ordered data, compared to others types such as selection which would have O(n^2).
- Descending order, will enable us to determine the behaviour under the worse case. In these tests shell sort algorithms will be superior as they order the data to a certain extent, enabling us to distinguish between insertion and shell sorts.
- Random lists of a large amount of numbers will be good to use to test for the efficiency of shell sorting algorithm to see the time take for the sort to occur

Performance Analysis

In our performance analysis, we measured how each program's execution time varied as the size and initial sortedness of the input varied. We used the following kinds of input ascending, descending and random. We also used different values associated with the keys to determine if the algorithm was stable or not.

We used these test cases because we are able to test the best/worst case, as well as general effectiveness.

Because of the way timing works on Unix/Linux, it was necessary to repeat the same test multiple times and take the average of three as the time, in order to limit the effect of any outliers.

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 there was no need to check the output.

We also investigated the stability of the sorting programs by referencing the original data set with the sorted set in order to determine that for the same keys the associated data preserves ordering

Experimental Results

Correctness Experiments

An example of a test case and the results of that test is “5, 4, 3, 2 , 1”

On all of our test cases, the expected “1,2,3,4,5” was obtained

Performance Experiments

For Program A, we observed that

- Preserves order
- Presorted data is sorted much quicker.
- Worse case with reverse data

These observations indicate that the algorithm underlying the program … *has the following characteristics:*

- Stable
- Best case of O(n)

For Program B, we observed that:

- Preserves order
- Relative to sortA much quicker.
- Approximately the same amount of time for 10000 numbers
- Random is much slower, relative to ascending/ descending, for 100 000 elements.
- Ascending /descending is roughly the same

These observations indicate that the algorithm underlying the program … *has the following characteristics:*

- Stable
- Works well for sorted data, regardless of the right way or not

Conclusions

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

- ProgramA implements the *insertion* sorting algorithm
- ProgramB implements the *shell* sorting algorithm

Experimental results: (seed: test, no 10000)

	Ascending	Descending	Random	Stability
sortA	0.16 + 0.03	0.73 + 0.04	0.56 + 0.03	-
sortB	0.04 +0.03	0.01 + 0.03	0.01 + 0.02	-

	Ascending	Descending	Random	Stability
sortA	0.152 + 0.04	0.71 + 0.03	0.58 + 0.03	-
sortB	0.01+ 0.2	0.01 + 0,02	0.01 + 0.02	-

	Ascending	Descending	Random	Stability
sortA	0.15 + 0.04	0.76 + 0	0.56 + 0.4	-
sortB	0.01 + 0.02	0.01 + 0.03	0.01 + 0.02	-

	Ascending	Descending	Random	Stability
sortA	0.15 + 0.04	0.74 + 0.02	0.57 + 0.03	Yes
sortB	0.01 + 0.02	0.01 + 0.03	0.01 + 0.02	Yes

Sort b will need to be repeated with more sorting (with 900000 numbers):

1. Random: 0.90 + 1.48
2. Ascending: 0.60 + 1.34
3. Descending: 0.68 + 1.37