COMP2521 Sort Detective Lab Report

By Hansen Liu, Shiyu Du.

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
measure their performance over a range of inputs:

Correctness Analysis

To determine correctness, we tested each program on the following kinds of input :
1. Partially ordered numbers(the last number in the first and the rest numbers are in ordered) with/without duplicates.
2. 50,000 random number with/without duplicates.
3. 50,000 reversed number with/without duplicates.
4. 50,000 ordered number with/without duplicates.
5. only 1 number in the list.

We chose these inputs because

These different kinds of input include all the circumstances. If a sorting program can pass these input, then its correctness can probably be guaranteed.

Performance Analysis

In our performance analysis, we measured how each program's execution time varied as the size and initial soreness of the input varied. We used the following kinds of input …
1. The lines of duplicate numbers with different marks behind.
2. Partially ordered numbers(the last number in the first and the rest numbers are in ordered)
3. 50,000 100,000 500,000 random number
4. 50,000 100,000 500,000 reversed number
5. 50,000 100,000 500,000 ordered number

We used these test cases because
1. The duplicate numbers with marks behind help us to determine the stability of the sort which eliminates about half of the choices.

2. Partially ordered numbers help us determine the adaptiveness of the sorting algorithm. (e.g. partially ordered numbers should decrease the run time of insertion sort but this does not apply to bubble sort.)
3. Random numbers with different size help to determine the bit O notation of the sorting algorithm.
4. The reversed number and ordered number help to find the different situation in different sort algorithm which helps to determine the big O notation.

Because of the way the timing works on Unix/Linux, it was necessary to repeat the same test multiple times to make sure that the data are reliable.

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 using inputs with duplicate numbers followed by different marks behind.
We also investigated any other relevant properties such as adaptiveness—to determine if the algorithm benefits from the presorted ness in the input sequence.

Experimental Results

Correctness Experiments

An example of a test case and the results of that test is used sorting algorithm to sort with different cases then put output to a .txt file, then use same input sort with the sorting in the system, diff with each other.
On all of our test cases,
SortA and SortB are correct sorting algorithm.

Performance Experiments:
For Program A, we observed that
The duplicates test shows that the program is not stable since it produces a different output each time.
If a number of inputs are the same, the run time of the program would be relevantly long if the input is randomly disordered. It takes less time sorting the reversed input and takes the least time sorting the partially ordered input.
The runtime of the program increases dramatically as the amount of input increase. The graph between amount of input and runtime looks similar to $y=X^2$

These observations indicate that the algorithm has the following characteristics
1. This sorting algorithm is not stable
2. This sorting algorithm is adaptive.
3. The run time of this algorithm has a big O notation of $O(n^2)$ sorting the random

numbers.

Since the sort is not stable. It can't be bubble sort with early exit or insertion sort or merge sort. Since it the run-time of the program is n^2, it can only be bubble sort or selection sort. Since the program takes less time sorting the reversed ordered input than random input, it can't be any kind of bubble sort. Hence, the only possibility of this sorting program is selection sort.

For Program B, we observed that

The duplicates test shows that the program is stable since it produces the same output each time.
If the size of inputs is the same, the run time of the program would be relevantly long if the input is randomly disordered. It takes less time sorting the reversed input and takes the least time sorting the partially ordered input.
The runtime of the program increases dramatically as the amount of input increase. The graph between amount of input and runtime looks similar to y=xlogx

These observations indicate that the algorithm has the following characteristics
1. This sorting algorithm is stable
2. This sorting algorithm is adaptive.
3. The run time of this algorithm has a big O notation of O(nlog(n)) sorting the random numbers.

Since the sorting algorithm is stable, there are only three possibilities of sorting algorithms left: bubble sort with early exit, insertion and merge sort. Besides, the big O notation is nlogn instead of n^2, it can only be merge sort.

Conclusions

On the basis of our experiments and our analysis above, we believe that
ProgramA implements the selection sorting algorithm
ProgramB implements the merge sorting algorithm

Appendix
Any large tables of data that you want to present

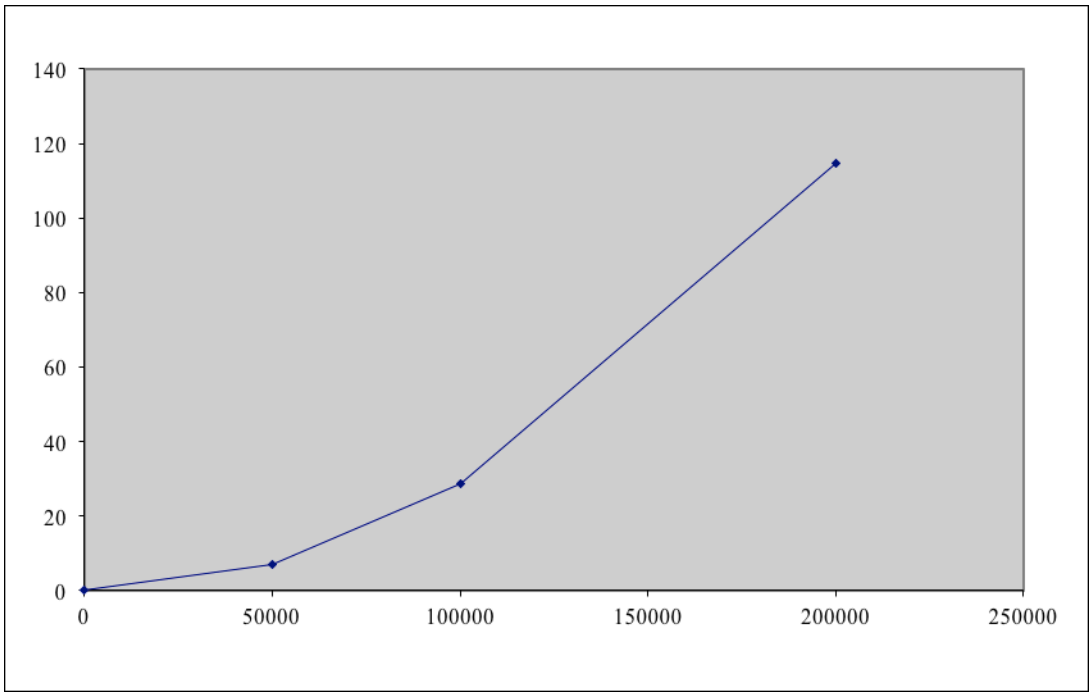
Program A:

Small Input (50000) sortA						
	First test	Second test	Third test	Fourth test	Fifth test	average time
Random Number	7.016	7.184	7.088	7.128	7.124	7.108
Reverse Number	5.64	5.484	5.464	5.492	5.536	5.5232
Partial Ordered Number	3.048	3.012	3.044	3.04	2.985	3.0258
Is it stable?	no					
adaptive?	yes					

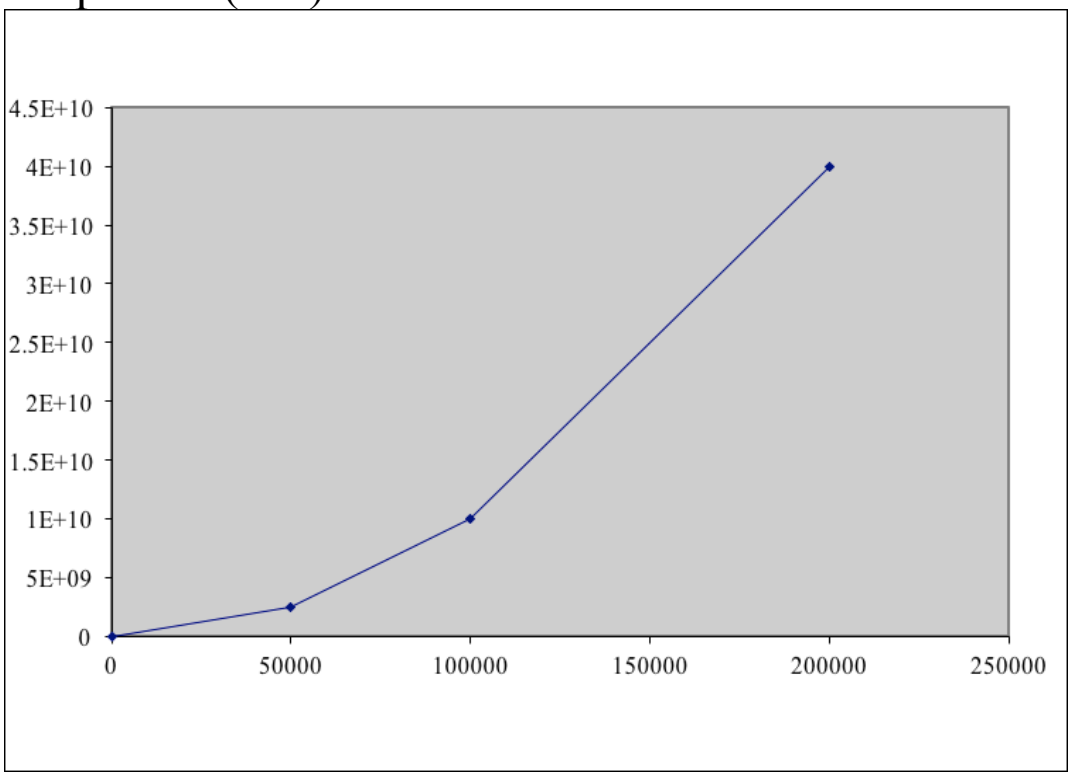
medium Input (100000) sortA						
	First test	Second test	Third test	Fourth test	Fifth test	average time
Random Number	28.540	28.560	28.812	28.788	28.592	28.658
Reverse Number	21.904	22.640	24.200	24.340	23.960	23.409
Partial Ordered Number	12.758	13.848	11.772	12.724	11.712	12.563

Large Input (500000) sortA						
	First test	Second test	Third test	Fourth test	Fifth test	average time
Random Number	114.436	114.546	114.128	115.124	114.344	114.5156
Reverse Number	88.16	87.792	90.778	87.564	88.284	88.5156
Partial Ordered Number	46.508	46.648	46.468	46.552	46.652	46.5656

Graph of sort A



Graph of O(n^2)



Program B:

Small Input (50000) sortB						
	First test	Second test	Third test	Fourth test	Fifth test	average time
Random Number	0.044	0.048	0.048	0.044	0.048	0.0464
Reverse Number	0.028	0.032	0.04	0.04	0.036	0.0352
Partial Ordered Number	0.028	0.024	0.032	0.032	0.032	0.0296
Is it stable?	yes					
adaptive?	yes					

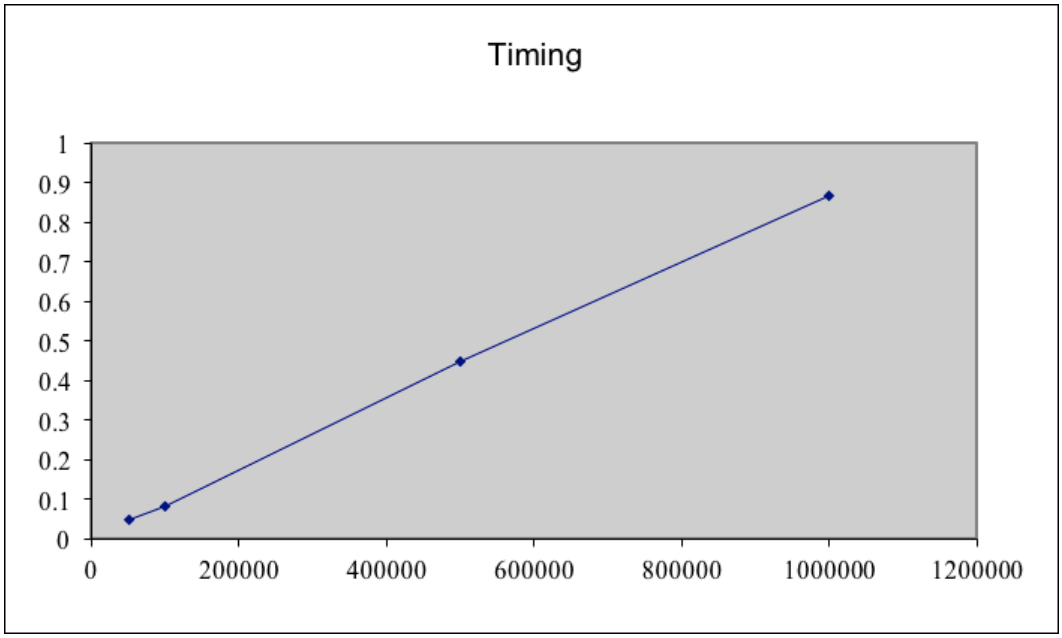
medium Input (100000) sortB						
	First test	Second test	Third test	Fourth test	Fifth test	average time
Random Number	0.084	0.081	0.092	0.088	0.072	0.083
Reverse Number	0.06	0.064	0.068	0.064	0.064	0.064
Partial Ordered Number	0.036	0.044	0.048	0.048	0.04	0.043

medium Input (100000) sortB						
	First test	Second test	Third test	Fourth test	Fifth test	average time
Random Number	0.084	0.081	0.092	0.088	0.072	0.083
Reverse Number	0.06	0.064	0.068	0.064	0.064	0.064
Partial Ordered Number	0.036	0.044	0.048	0.048	0.04	0.043

Large Input (500000) sortB						
	First test	Second test	Third test	Fourth test	Fifth test	average time
Random Number	0.400	0.472	0.436	0.46	0.472	0.448
Reverse Number	0.304	0.252	0.312	0.304	0.288	0.292
Partial Ordered Number	0.12	0.148	0.16	0.136	0.148	0.142

Larger Input (1000000) sortB						
	First test	Second test	Third test	Fourth test	Fifth test	average time
Random Number	0.892	0.932	0.844	0.832	0.84	0.868
Reverse Number	0.588	0.592	0.584	0.616	0.6	0.596
Partial Ordered Number	0.276	0.268	0.276	0.268	0.34	0.2856

Graph of sort B



Graph of O(nlogn)