COMP2521 Sort Detective Lab Report

by Zheng Luo (z5206267)

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

Experimental Design

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:

1. Ascending(sorted),
2. reversing,
3. completely randomised.

We used these test cases because:

1. By sorted number sets, the adaptiveness of sorting method can be determined, since non-adaptive functions will cost much more time than adaptive function, as which does not need the sorting operation when it found the numbers are sorted already.
2. Reversing number sets can expose the worst-case scenario for most of the algorithms, hence the worst time complicity can be roughly determined by checking the time difference between each set.
3. Completely randomised number sets can give an average time performance, which can solidify the assumptions deduced from other sets when comparing with one another.

Because of the way timing works on Unix/Linux, it was necessary to repeat the same test multiple times since the machine running time will slightly vary due to machine performance, in order to minimise the uncertainty, the same test should be run multiple times, and use their average time as reference.

Experimental Results

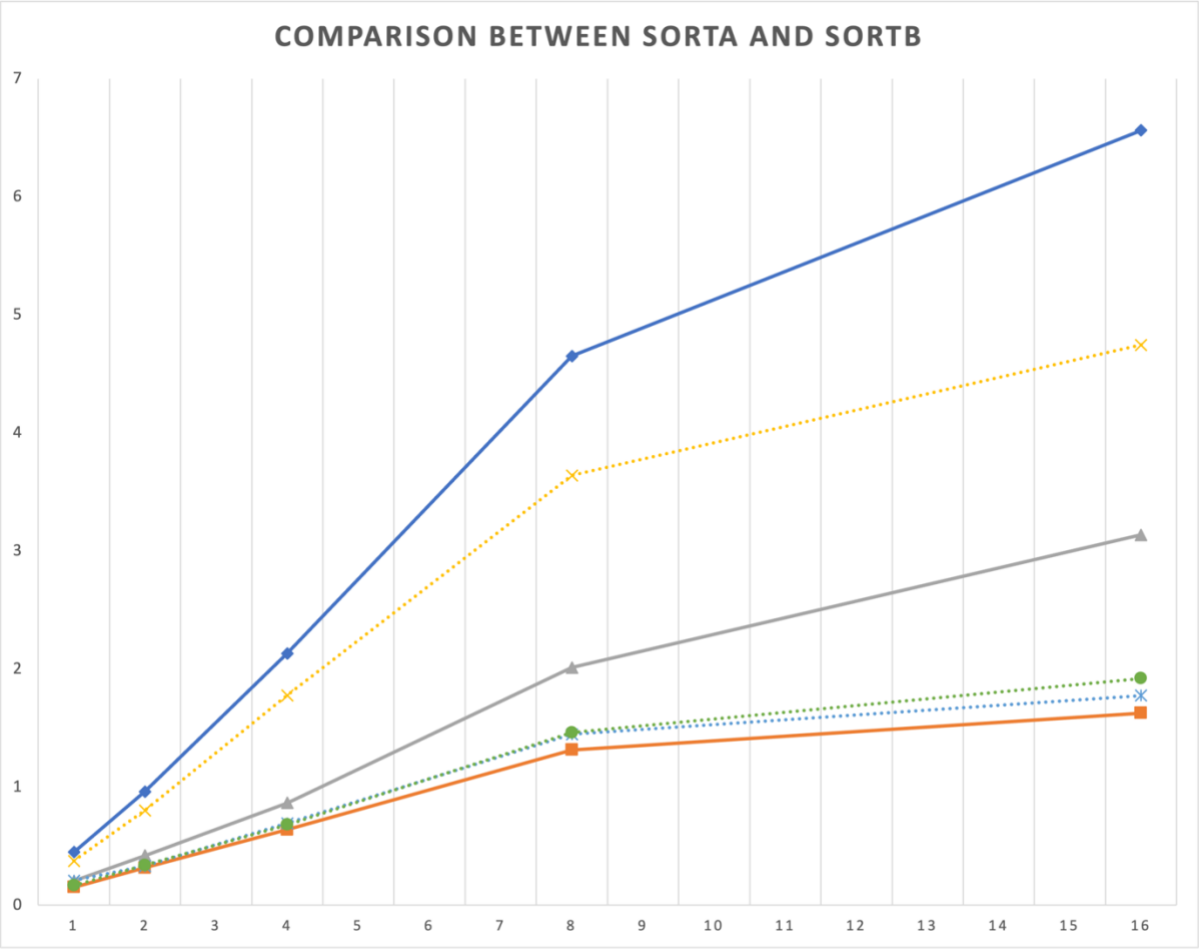

Figure 1 Comparison between sortA and sortB

The solid lines represent the time performance for program "sortA", and dotted lines represent for "sort B", and these are the average data time for 1, 2, 4, 8, 16 million data sets. The most top two lines are representing random as expected, and the middle two are indicating reversed, and sorted are the lowest 2 lines.

Program A cost more time than program B for sorting random and reversed data sets, but sorted case. Program A has the advantage when dealing with sorted data sets, hence program B can be deduced as non-adaptive, and adaptive for program A. However, there is some possibility of both programs are adaptive or non-adaptive since they are very close to each other.

Both program A and program B have same timing pattern of equal or less to $0(n)$, which reduced the searching time cost when they searched 16 million sets, the time complicity can be ranged within $0(n)$ and $0(nlog(n))$. The pattern similarity also indicates they have the

same time complexity for both best and worst case scenarios. Base on these information, the selections for these two programs can be narrowed to merge sort and heap sort, since they both have $0(nlog(n))$ for best and worst case.

Another key discovery is that the cost of program B sorting reversely have similar cost as sorting sorted sets. The fixdown function cost $O(\log(n))$ in heap sort, the reversed number sets do not require move the larger number to the rear, since they are already there, so the cost will significantly reduced. Therefore, program B is heap sort, and program A is merge sort.

Conclusions

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

- sortA implements the *merge* sorting algorithm
- sortB implements the *heap* sorting algorithm

Appendix

	Size 1000000			Size 2000000			Size 4000000			Size 8000000			Size 16000000		
SortA	Random	Sorted	Reversed	Random	Sorted	Reversed	Random	Sorted	Reversed	Random	Sorted	Reversed	Random	Sorted	Reversed
Test1 A	0.54	0.15	0.21	0.99	0.36	0.48	2.21	0.64	0.89	4.47	1.31	1.97	6.39	1.58	3.19
Test2 A	0.42	0.15	0.17	0.94	0.3	0.37	2.14	0.65	0.86	4.88	1.37	1.97	7.1	1.66	3.15
Test3 A	0.42	0.14	0.22	0.94	0.29	0.38	2.18	0.64	0.88	4.82	1.36	1.96	6.27	1.56	2.92
Test4 A	0.43	0.17	0.19	0.99	0.3	0.38	2.06	0.66	0.86	4.5	1.32	2.05	6.64	1.68	3.22
Test5 A	0.44	0.15	0.24	0.95	0.34	0.49	2.08	0.61	0.84	4.59	1.22	2.12	6.42	1.66	3.21
Test1 B	0.36	0.15	0.17	0.82	0.32	0.34	1.65	0.66	0.69	3.82	1.48	1.35	4.58	1.75	1.91
Test2 B	0.35	0.21	0.17	0.79	0.33	0.33	1.85	0.73	0.67	3.51	1.35	1.52	4.81	1.82	1.87
Test3 B	0.34	0.16	0.16	0.76	0.37	0.37	1.89	0.71	0.68	3.73	1.47	1.62	4.41	1.69	1.94
Test4 B	0.47	0.22	0.18	0.78	0.32	0.32	1.74	0.66	0.67	3.6	1.52	1.4	4.84	1.83	1.97
Test5 B	0.36	0.3	0.16	0.86	0.32	0.34	1.76	0.71	0.7	3.54	1.42	1.42	5.09	1.79	1.92

Figure 2 raw data for sorting program

	Random	Sorted	Reversed	
1	0.45	0.152	0.206	
2	0.962	0.318	0.42	
4	2.134	0.64	0.866	A
8	4.652	1.316	2.014	
16	6.564	1.628	3.138	
1	0.376	0.208	0.168	
2	0.802	0.332	0.34	
4	1.778	0.694	0.682	B
8	3.64	1.448	1.462	
16	4.746	1.776	1.922	

Figure 3 Average data sets for sorting program