COMP2521 Sort Detective Lab Report

by Liuheng ZHOU(z5271696)

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 random, sorted, reversed and my datasets.

In my datasets, mydata.txt is all 5 but second last one is 1, mydataA.txt is sorted, mydataR.txt is random.

We used these test cases because they can highlight the strengths and weaknesses of different sorting algorithms, helping us identify the underlying algorithm based on its performance characteristics. In addition, my datasets can find out stable or not and deeply analyze.

Because of the way timing works on Unix/Linux, it was necessary to repeat the same test multiple times to get a consistent measure of the program's performance.

We also investigated the stability of the sorting programs by testing them with inputs that had duplicate keys. This allowed us to determine if the relative order of equal keys was preserved after sorting.

Algorithm	Sorted	Reversed	Random	Stable
Bubble sort	O(n)	O(n^2)	O(n^2)	Yes
Insertion sort	O(n)	O(n^2)	O(n^2)	Yes
Selection sort	O(n^2)	O(n^2)	O(n^2)	No
Merge sort	O(n*log(n))	O(n*log(n))	O(n*log(n))	Yes
Naive quicksort	O(n^2)	O(n^2)	O(n*log(n))	No
Median-of-three quicksort	O(n^2)	O(n^2)	O(n*log(n))	No
Randomised quicksort	O(n^2)	O(n^2)	O(n*log(n))	No
Bogosort	O(n!)	O(n!)	O(n!)	No

Experimental Results

For Program A (sortA), we observed that the time taken for random and reversed inputs increased roughly quadratically in O(n^2), while the time taken for sorted inputs was consistently very low in O(n).

Additionally, sortA was found to be stable.

Also, mydata.txt tests shows that Bubble Sort with early exit: It will take several passes to move the 1 to its correct position near the beginning. The early exit won't trigger until the entire list is sorted. This can explain the significant time taken. Insertion Sort: It will quickly place the 1 in its correct position. Since the rest of the numbers are the same, it will move through them without making any swaps. However, it will still take time to traverse the list and check each element.

These observations indicate that the algorithm underlying sortA has the characteristics of the "Bubble Sort with Early Exit" algorithm.

For Program B (sortB), we observed that the time taken for random inputs was consistently very low, while the time taken for sorted and reversed inputs increased roughly quadratically. Additionally, sortB was found to be unstable.
These observations indicate that the algorithm underlying sortB has the characteristics of the "Naive Quicksort" algorithm.

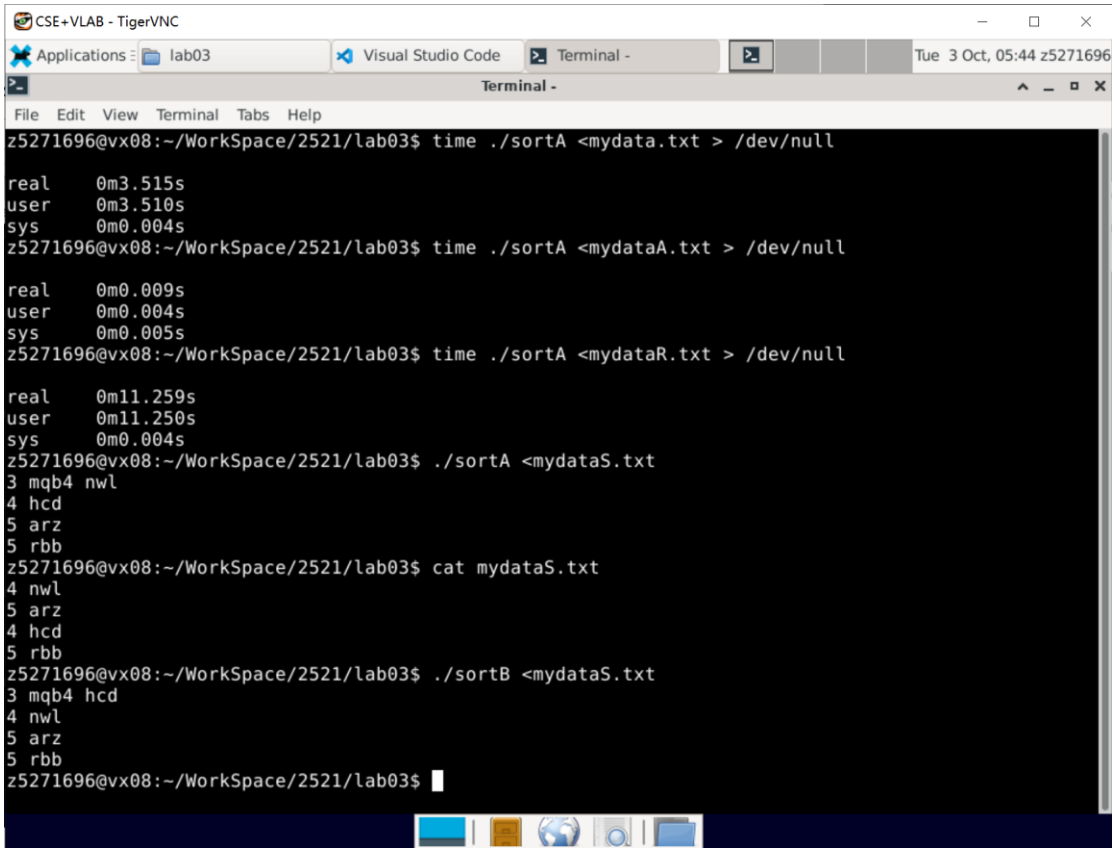
Conclusions

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

- sortA implements the *Bubble* sorting algorithm
- sortB implements the *Naive Quicksort* sorting algorithm

Appendix

Any large tables of data that you want to present ...



Test Case	Input Type	Input Size	sortA Time (Average)	sortB Time (Average)
1	Random	10000	0.42	0
2	Sorted	10000	0	0.13
3	Reversed	10000	0.48	0.13
4	Random	25000	2.72	0
5	Sorted	25000	0	0.8
6	Reversed	25000	2.83	0.83

7	Random	50000	11.58	0.01
8	Sorted	50000	0	3.13
9	Reversed	50000	11.61	3.35
10	Random	75000	25.62	0.02
11	Sorted	75000	0.01	7.15
12	Reversed	75000	25.56	7.64
13	Random	100000	46.14	0.03
14	Sorted	100000	0.01	12.51
15	Reversed	100000	50.97	13.58
16	Random	250000	NULL	0.07
17	Sorted	250000	NULL	Command terminated
18	Reversed	250000	NULL	Command terminated
19	Random	500000	NULL	0.18
20	Sorted	500000	NULL	NULL
21	Reversed	500000	NULL	NULL
22	Random	1000000	NULL	0.44
23	Sorted	1000000	NULL	NULL
24	Reversed	1000000	NULL	NULL