COMP2521 Sort Detective Lab Report

by Alexander Lai

Experimental Design

I measure how each program's execution time varied as the size and initial sortedness of the input varied. We used the these types of input:

Input with varying sizes (10 000 - 160 000 items) that were sorted in different ways (random, ascending order, descending order).

I used these test cases to gauge the time complexity of the sort programs we were given so that we can compare them to known time complexities to figure out the sorting algorithms.

To ensure accuracy and reliability, we repeated each timing test case 5 times.

I also investigated the stability of the sorting programs by giving the programs a list of inputs which contained equal items with different keys. If the outputs and if the order of the keys were preserved, the algorithm is considered stable.

Experimental Results

For Program A, we observed that it is a stable sorting algorithm as shown by the results in the appendix as the order of the keys were preserved after sorting. This means that the sorting algorithm must be either **bubble sort, insertion sort or merge sort**, as those are the only stable sort algorithms.

Looking at the timing data for all the inputs (ascending, random and descending) we can see that the timings for each type of input have relatively similar times. This means that the worst case, best case and average case have generally the same time complexity. Therefore, Program A is a **MERGE SORT.**

For Program B, we observed that it is an unstable sorting algorithm as shown by the results in the appendix as the order of the keys were not preserved after sorting. This means the sorting algorithms are either **selection sort, naive quicksort, randomised quicksort, median of three quicksort or bogosort,** as they are unstable sort algorithms.

It is clear from the timing data for inputs which are sorted that the algorithm underlying the program cannot be selection sort or bogosort as both have best case time complexities larger than O(n) time. Furthermore it cannot be a naive quicksort either as the time complexity would

be O(n^2) for lists that are in ascending order. This leaves **randomised quicksort** or **median of three quicksort** as the only options as they have the same time complexity for average and best cases.

Because the decreasing and ascending average times are very similar in the sort B algorithm, we can see that it follows the common expected trends of the **Median-of-three quicksort ,** the sort B algorithm implements the median-of-three quicksort algorithm.

Conclusions

Based on our experiments and our analysis above, we believe that

- sortA implements the ***merge sort*** algorithm
- sortB implements the ***median of three quicksort*** sorting algorithm

Appendix

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

Type of Sort	**Time Complexity**	**Stability**
Bubble Sort	Worst - $O(n^2)$ Best - O(n) - occurs when input is already sorted, will terminate after one loop. Average - $O(n^2)$	Stable sorting algorithm
Insertion Sort	Worst - $O(n^2)$ Best - O(n) - occurs when already sorted Average - $O(n^2)$	Stable sorting algorithm
Selection Sort	Worst - $O(n^2)$ Best - $O(n^2)$ Average - $O(n^2)$	Unstable sorting algorithm
Merge Sort	Worst - O(n log n) Best - O(n log n) Average - O(n log n)	Stable sorting algorithm
Naive Quicksort	Worst - $O(n^2)$ Best - O(n log n) Average - O(n log n)	Unstable sorting algorithm
Median-of-three quicksort	Worst - $O(n^2)$ Best - O(n log n)	Unstable sorting algorithm

	Average - O(n log n)	
Randomised Quicksort	Worst - O(n^2) Best - O(n log n) Average - O(n log n)	Unstable sorting algorithm
Bogosort	Worst - Unbounded Best - O(n) Average - O((n + 1)!)	Unstable sorting algorithm

SortA	AVG Time (s)
Ascending (Size)	
10000	0.16
20000	0.64
40000	2.54
80000	10.19
160000	40.74
Random (Size)	
10000	0.16
20000	0.64
40000	2.56
80000	10.20
160000	40.78
Descending (Size)	
10000	0.15
20000	0.61
40000	2.44
80000	9.80

160000	39.14

SortB	AVG Time (s)
Ascending (Size)	
500,000	0.08
2,000,000	0.39
5,000,000	1.00
10,000,000	2.06
Random (Size)	
500,000	0.17
2,000,000	0.86
5,000,000	2.33
10,000,000	4.90
Descending (Size)	
500,000	0.09
2,000,000	0.40
5,000,000	0.98
10,000,000	2.04

Testing Stability of the Sorting Algorithms

Giving the two programs the following input:

2 aaa
2 bbb
2 ccc
2 ddd
2 eee
3 aaa
3 bbb
3 ccc
3 ddd
3 eee
1 aaa
1 bbb
1 ccc
1 ddd
1 eee

Sort A Produced:

```
1 aaa
1 bbb
1 ccc
1 ddd
1 eee
2 aaa
2 bbb
2 ccc
2 ddd
2 eee
3 aaa
3 bbb
3 ccc
3 ddd
3 eee
```

Sort B Produced:

```
1 bbb
1 aaa
1 ccc
1 eee
1 ddd
2 aaa
2 ccc
2 eee
2 bbb
```

```
2 ddd
3 ccc
3 bbb
3 aaa
3 ddd
3 eee
```