COMP2521 Sort Detective Lab Report

by Liren Ding (z5369144)

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 10000 20000 40000 80000 160000 in random sorted reversed cases

We used these test cases because test best case, average cases, and worst case

Because of the way timing works on Unix/Linux, it was necessary to repeat the same test multiple times to ensure the time complexity

We also investigated the stability of the sorting programs by `./sortA < mydata > sortedA`

Experimental Results

For Program A, we observed that sorted data is always sorted fastest. Reversed data is always sorted slowest, and random data is always sorted few faster than reversed data.

```
 === Testing for random input, size 10000 ===
0.44
0.44
0.43
0.44
0.44

=== Testing for sorted input, size 10000 ===
0.00
0.00
0.00
0.00
0.00
```

```
=== Testing for reversed input, size 10000 ===
0.49
0.49
0.49
0.48
0.49
```

These observations indicate that the algorithm underlying the program is bubble sort has the following characteristics:

When data is sorted, time complexity is O(1), from the nswap == 0 condition.

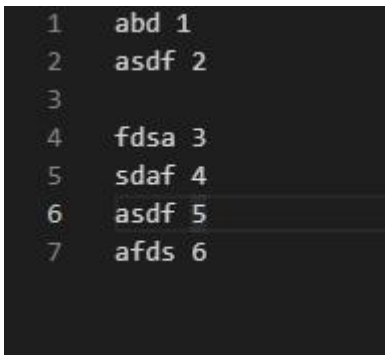
For reversed data, it should keep going bubble sorting till the final data. And if we look at random case now, we will discover bubble sort will not execute till the final data, random case is always be better than reversed case(the worst case)

```
1  void bubbleSort(int a[], int lo, int hi) {
2     int i, j, nswaps;
3     for (i = hi; i > lo; i--) {
4        nswaps = 0;
5        for (j = lo; j < i; j++) {
6           if (less(a[j + 1], a[j])) {
7              swap(a[j + 1], a[j]);
8              nswaps++;
9           }
10       }
11       if (nswaps == 0) break;
12    }
13 }
```

Stability test:

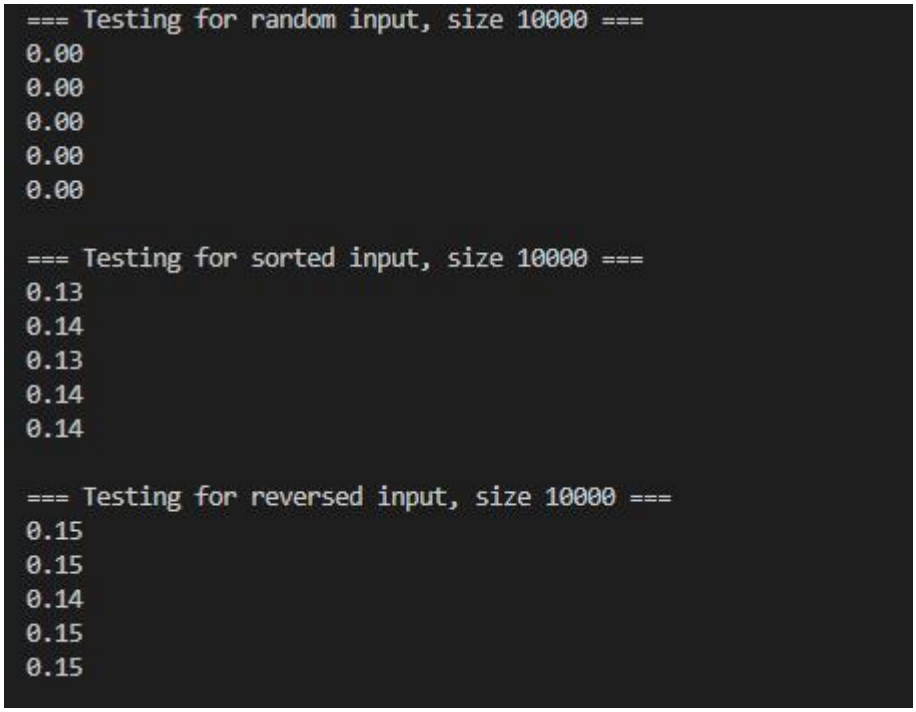
```
1   abd 1
2   asdf 2
3
4   fdsa 3
5   sdaf 4
6   asdf 5
7   afds 6
```

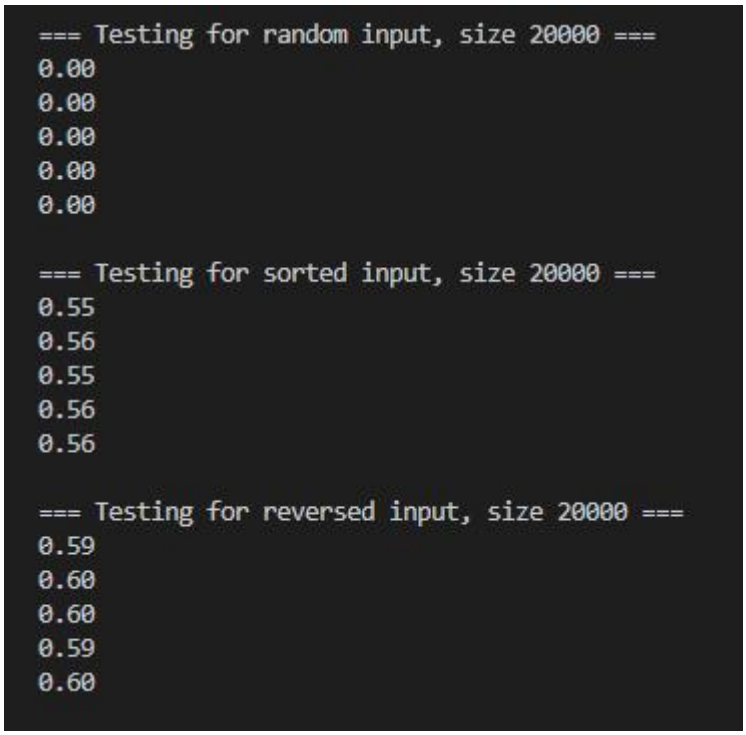
After sorted



Unstable

For Program B, we observed that random data is sorted fastest. Sorted data is always few faster than reversed data



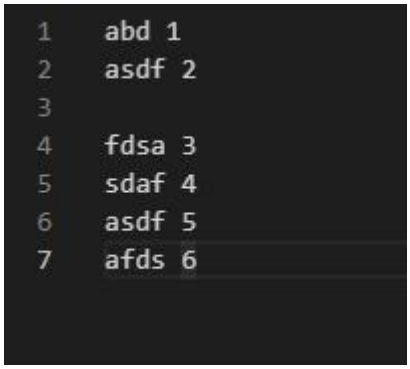


These observations indicate that the algorithm underlying the program is quick sort, which has the following characteristics

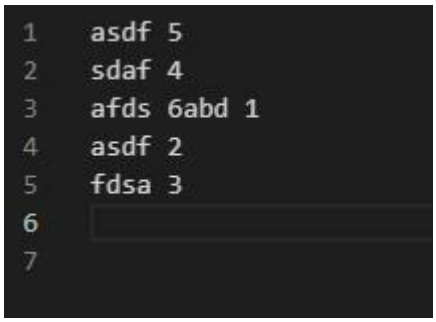
Worst case: The original sequence is in order, and the reference value is chosen as the maximum or minimum value each time. At this time, it is degraded to bubble sorting, and the time complexity is O

Best case: A reference value that splits the array exactly in half each time it is selected

Stability test:



After sorted:



Unstable

Conclusions

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

- sortA implements the *bubble* sorting algorithm
- sortB implements the *quick* sorting algorithm