COMP2521 Sort Detective Lab Report

（z5306254）Sicong GAO

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) Some size of random numbers (size: small to big).
2) Some size of numbers in ascending order (size: small to big).
3) Some size of numbers in descending order (size: small to big).
4) A little size of random numbers, but include duplicate, like {10 a, 10 b, 5}.

We used these test cases because:

The first test set aims to test the time complexity of the sorting algorithm, because when the scale becomes larger, the time complexity becomes stable and mathematical.

The second test aims to test the adaptive of the sorting algorithm, when we already sorted the set, what time complexity for an algorithm going to O(n) or just same as the random one?

The third test aims to check some worse case.

The fourth test aims to test the stability of a sorting algorithm, we will observe whether the same items order will be changed.

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

We investigated the stability of the sorting programs by test 4.

We also investigated adaptive by test 2 and the time complexity by test 1.

Experimental Results

For Program A, we observed that, when we do test4, which include some duplicate items, the program A didn't change the same items order, so it's stable. And then we also use test1 and 2. It display the time complexity is O(n^2), when we do with the random numbers. And when we do with the sorted numbers which are in ascending order, then time is very close to 0.

So, we can know that it is adaptive, and time is O(n^2), then the possible is bubble and insertion sort.

However, by comparing the results of executing programs A and B, we can clearly know that in the worst case, program B is always faster than program A, so we can know that program A is bubble sort, because in the reverse case, the bubble sort must swap every item in each iteration, but in the quick sort, we just do it until i == j, and then swap it once. So, in a worse case, quick sort will be faster than bubble sort.

For program B, we observe that when we do test 4, program B changes the same element order, so we can clearly know that it is unstable, then the possible algorithm of B becomes to selection, quick or Bogosort.

Then for test2, when we do the test with sorted and reversed numbers, their time is very close. And it is much longer than random numbers. So, we can know that it is not adaptive, then exclude selection sort, and Bogosort. Therefore, it is quick sort. In addition, because sorted and

reverse number test time are similar and take longer, it is a Naive quick sort. Because we must use each item as a pivot once.

And for the test1, the outcome for the random size time complexity close to O(nlogn), but for the sorted and reverse sizes, it’s increase to O(n^2). Thus, it’s confirms our hypothesis.

These observations indicate that the algorithm underlying

Program A: it is stable, adaptive and always is O(n^2).

Program B: it is unstable and unadaptive. And for the random it is O(nlogn),

For the sorted and reverse it is O(n^2).

Conclusions

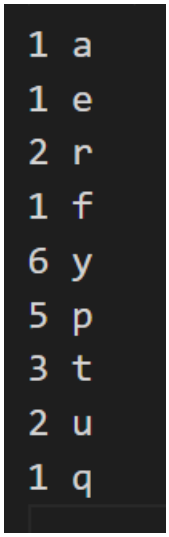
On the basis of our experiments and our analysis above, we believe that

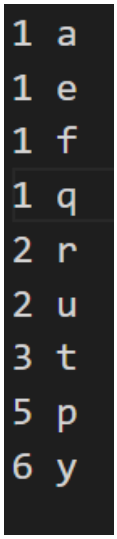
- sortA implements the bubble sorting algorithm
- sortB implements the Naive quick sorting algorithm

Appendix

For program A:

1) stable test:
   input: {10a, 10b, 5}
   output: {5, 10a, 10b}





Conclusion: Doesn't change same items order.
Stable sort

2) Time complexity:

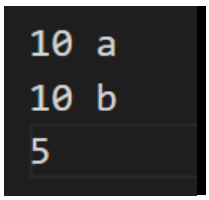
size	Random time	Sorted time	Reverse time
10000	0.42	0.00	0.44
20000	1.65	0.00	1.72
40000	6.82	0.00	6.98
80000	28.36	0.01	28.35
160000	112.82	0.02	113.68

In the worse case, since bubble sort need swap everytime, but quick sort just swap once, So, bubble sort will be always slower than quick sort, just compare the worse case.
Conclusion: bubble sort and adaptive.

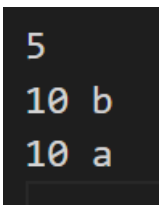
For program B:

1) Stable test:

Input:



output:



Conclusion: Unstable sort

2) Time complexity:

size	Random time	Sorted time	Reverse time
10000	0.00	0.14	0.16

20000	0.00	0.59	0.58
40000	0.01	2.07	2.17
80000	0.03	8.25	8.68
160000	0.07	32.56	34.61

Conclusion: Not Adaptive, and naive quick sort, choose pivot by the left most item.