Week 6 Experiment report

Zhengyue LIU z5036602

1.Experiment Design:

There are two aspects to our analysis:

- determine that the sort programs are correct, and
- measure their performance over a range of inputs.

The final goal of this experiment is to identify the algorithm used by the program.

1.1 Correctness Analysis

To determine correctness of the algorithm, we will test the algorithm with three sets of random numbers, each with 100000 elements, and ensure that it is sorted by using diff command**.**

1.2 Performance Analysis

In our performance analysis, we measured how each program's execution time varied as a function of the size and the initial sortedness of the input. The following kinds are tested to help identify the algorithm:

1. Any input with relative size will isolate the Pseudo-Bogo Sort from the rest.
2. Random numbers (to get average time complexity of the program)

Reason: this input will help us distinguish the shell algorithm with the rest of the algorithms. The shell algorithm can give O(n*logn) or O(n^4/3) or O(n^3/2), which is faster than the rest of the algorithms. The rest of algorithms will give O(n^2) on average.

3. Completely sorted sequence of numbers

Reason: this input will give best case for insertion sort and Bubble Sort with Early Exit. Bubble Sort with Early Exit will give O(n). Insertion sort will also give O(n). The Oblivious Bubble Sort and selection sort will still give O(n^2).

4. Repetitive key inputs with different attributes

Reason: this input will show us the stability of the algorithm. By seeing whether the attributes with the same key remain in the same order as in input, we can tell whether the program is a stable sort or not. Luckily, the selection and shell sort are unstable. It helps us to distinguish them from the rest. Combining with sorted sequence of key, we can easily identify the selection sort, because bubbles with no exit sort remain stable under the worst case.

5. Input 1 [8 7 6 5 4 3 2 1.abc 1. abcd], Input 2 [5 4 3 1.abc 1. abcd]

Reason: If the interval is 1, 8, then the input1 after sorting will show instability, but Input2 will not. Since, if the interval is 1, 8, then for Input1, at first pass, key number 1 with the attribute abcd will be switched with key number 8, and then when first pass ended, interval will be changed to 1. This is essentially just an insertion sort. Key number 1 with attribute abc will stop after key number 1 with attribute abcd. Essentially positions of the original key number 1 are swapped. For input2 there will be only one pass with interval 1, and key number 1 with attribute abc will move to front first, then the 1.abcd will be blocked after. If the interval is 1,4 (Power of four shell sort) then Input2 will also show instability for the same reason explained above.

2. Experiment result and analysis:

2.1 Correctness Experiments

We tested the algorithm with three sets of random numbers, each with 100000 elements, and ensured that it is sorted by using diff command. Additionally, we also tested both programs with descending sequence of number to ensure the program is sorting properly.

2.2 Performance Experiments

2.2.1 Random data test experiment results:

SortA and SortB are tested by sequence of random number of variant size. Then we constructed the detailed timing table (Appendix 4.1 & 4.2) and plotted the graph.

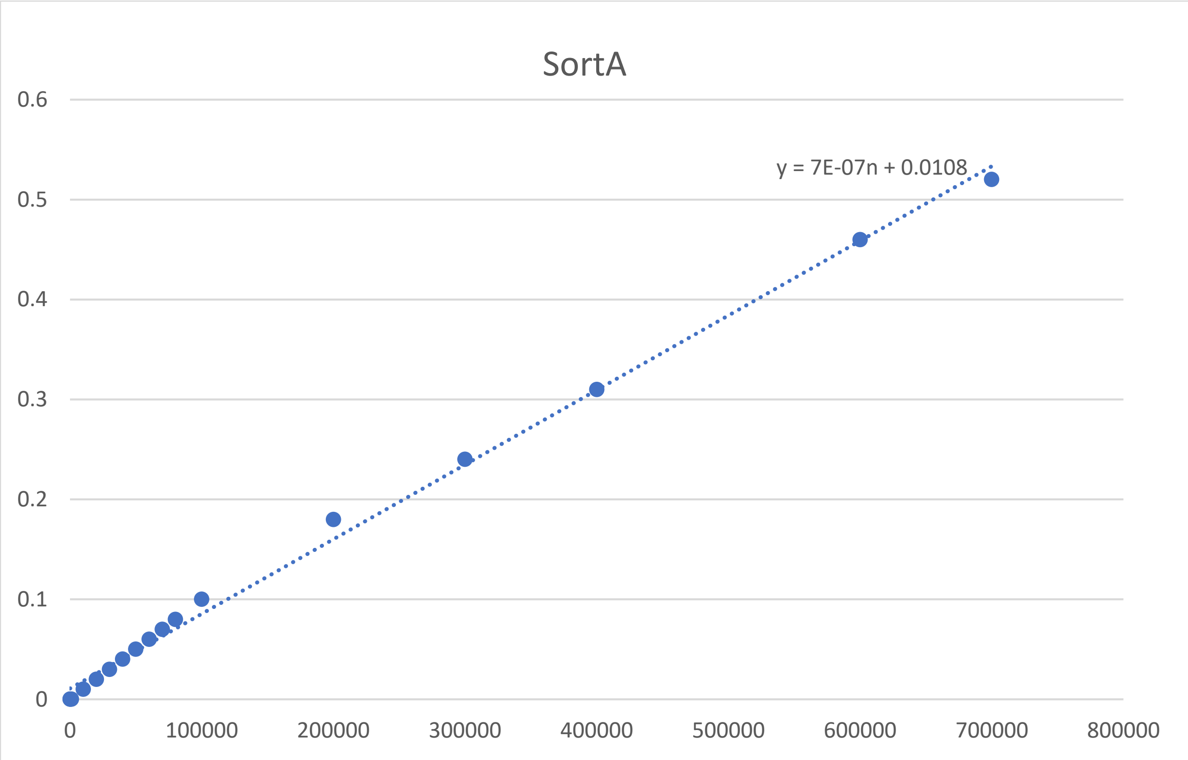


Fig. 1

Fig.1 shows that the complexity of SortA is little different from O(n), and very closed to O(nlogn). Hence, we can tell that the SortA is one of the shell algorithms.

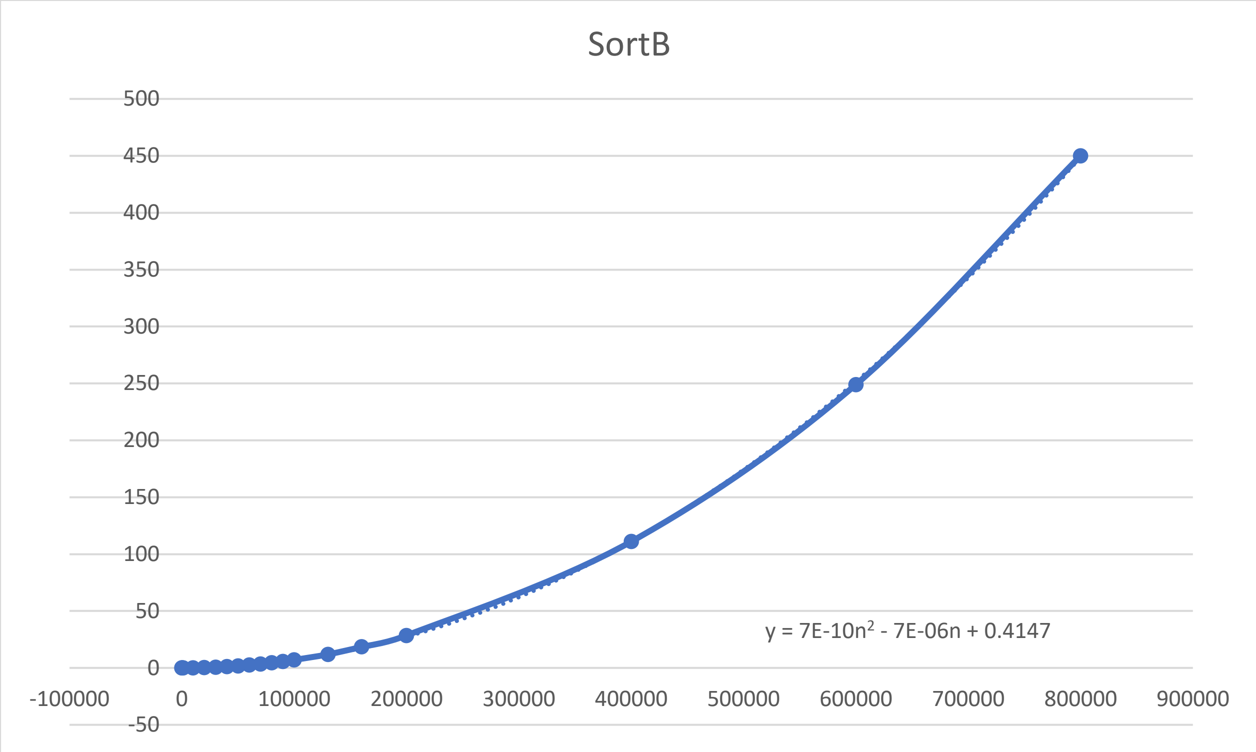


Fig. 2

Fig.2 shows that the time complexity of the sortB is O(n^2), indicating that the sortB is possibly an insertion sort, selection sort or bubble sort. From the graph, the sortB is more likely to be bubble sort because the insertion sort is little faster than bubble sort or selection sort (O(n^2)) on average case.

2.2.2 Completely sorted sequence of numbers (best case test) result:

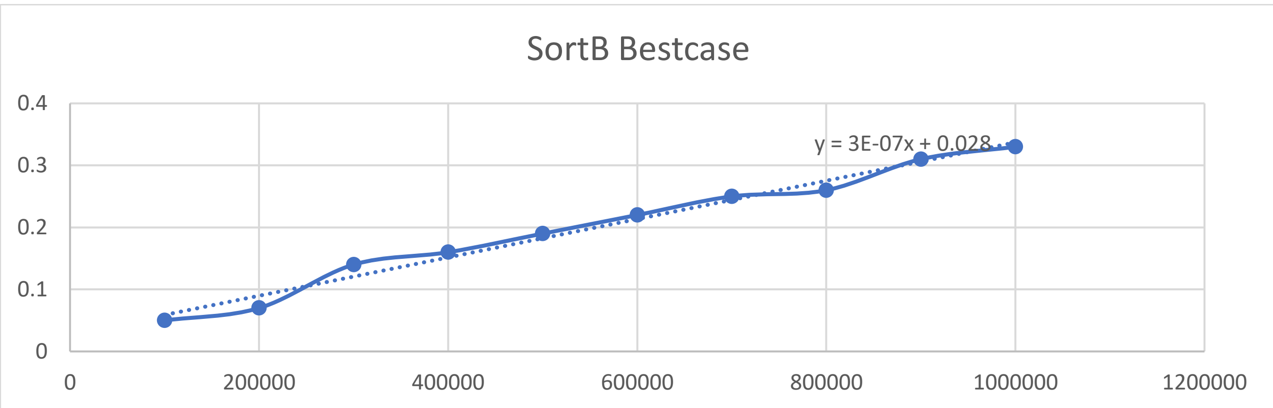


Fig.3

Fig.3 shows that the time complexity of the sortB at thebest case is O(n). So, the sortB possibly is an adaptive bubble or insertion sort.

2.2.2 Stability test experiment result:

SortA and SortB are tested by sequence of data with repetitive elements to check the stability of the sorting algorithm. The actual input data and output checking are attached in Appendices.

The output shows that the SortA is unstable while the SortB is stable. Hence, we can ensure that the SortA is a shell sort. Since the SortB is stable with average time complexity O(n^2), we can deduce that the SortB is either an insertion sort or a bubble sort.And, from the Fig.2,its more likely to be a bubble sort.

2.2.3 Input 1 [8 7 6 5 4 3 2 1.abc 1. abcd], Input 2 [5 4 3 1.abc 1. abcd] experiment result:

After sorting, input 1 becomes [1. abcd 1.abc 2 3 4 5 6 7 8], input 2 becomes [1.abc 1.abcd 3 4 5]. Hence, we can conclude that SortA is a Sedgewick shell sort.

3.Conclusions

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

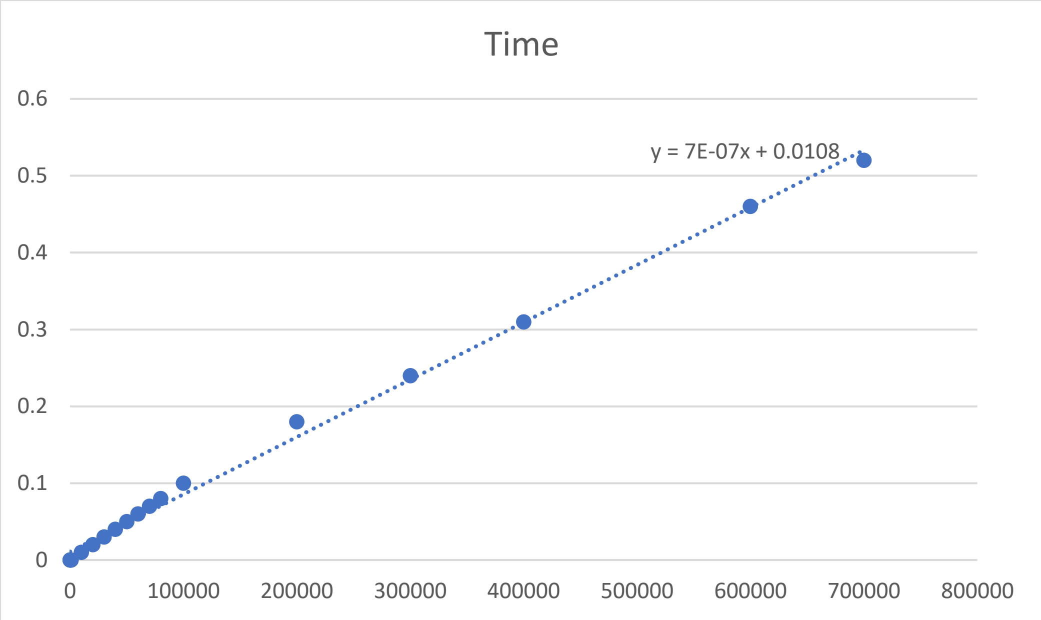
1. Sort A implements the Sedgewick shell sort algorithm, and

2. Sort B implements the bubble sort with early exit sorting algorithm.

Appendix:

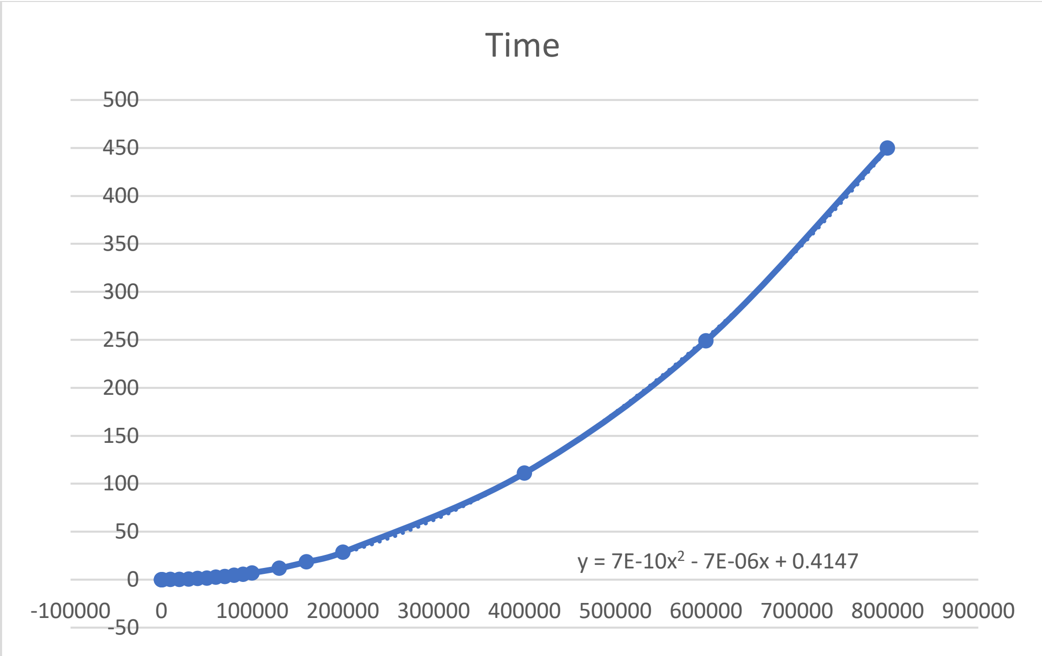
4.1 Detail timing of random data test on SortA

T(n)	Average	Measurements		
10	0	0	0	0
100	0	0	0	0
1000	0	0	0	0
10000	0.01	0.01	0.01	0.01
20000	0.02	0.02	0.02	0.02
30000	0.03	0.03	0.03	0.03
40000	0.04	0.04	0.04	0.04
50000	0.05	0.05	0.05	0.05
60000	0.06	0.06	0.06	0.06
70000	0.07	0.07	0.07	0.07
80000	0.08	0.08	0.08	0.08
100000	0.1	0.1	0.1	0.1
200000	0.18	0.18	0.18	0.17
300000	0.24	0.24	0.24	0.24
400000	0.31	0.31	0.31	0.31
600000	0.46	0.47	0.45	0.46
700000	0.52	0.52	0.54	0.5


Time
0.6
0.5
0.4
0.3
0.2
0.1
0
y = 7E-07x + 0.0108
0
100000
200000
300000
400000
500000
600000
700000
800000

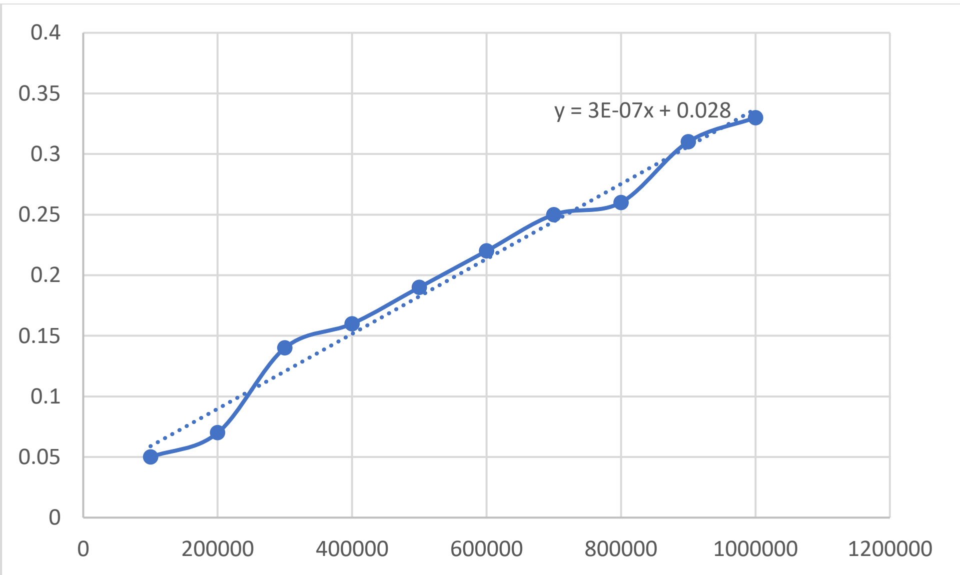
4.2 Detail timing of random data test on SortB

T(n)	Average	Measurements		
10	0	0	0	0
100	0	0	0	0
1000	0	0	0	0
10000	0.13	0.13	0.13	0.13
20000	0.35	0.35	0.35	0.35
30000	0.68	0.68	0.68	0.68
40000	1.16	1.18	1.14	1.16
50000	1.78	1.8	1.76	1.78
60000	2.54	2.51	2.54	2.57
70000	3.46	3.5	3.43	3.47
80000	4.6	4.71	4.58	4.5
90000	5.71	5.78	5.65	5.72
100000	7.06	7.03	7.07	7.1
130000	11.9	11.82	12.05	11.82
160000	18.49	18.34	18.23	18.9
200000	28.55	28.7	28.35	28.6
400000	111	111	113	110
600000	249	247	250	251
800000	450	440	443	450


Time
500
450
400
350
300
250
200
150
100
50
0
-50
-100000
0
100000
200000
300000
400000
500000
600000
700000
800000
900000
y = 7E-10x2 - 7E-06x + 0.4147

4.3 Detail timing of sorted data test on SortB

T(n)	Average	Measurements		
100000	0.05	0.05	0.05	0.05
200000	0.07	0.07	0.07	0.07
300000	0.14	0.14	0.14	0.13
400000	0.16	0.17	0.15	0.16
500000	0.19	0.19	0.19	0.2
600000	0.22	0.22	0.22	0.22
700000	0.25	0.24	0.25	0.25
800000	0.26	0.28	0.26	0.24
900000	0.31	0.31	0.31	0.3
1000000	0.33	0.33	0.33	0.33



4.4 Output checking on stability

Mydata	SortA	SortB
100 abc	1 ewrqewrqe	1 ewrqewrqe
100 def	3 a1	3 a1
100 fadfag	13 rew	13 rew
100 gdagdsa	21 rewq	21 rewq
100 ewrqewtwq	34 a5	34 a5
93 ewqreer	38 dfgds	38 dfgds
65 ere	62 rewq	62 rewq
1 ewrqewrqe	62 fdsae	62 rewqerw
92 reqw	62 rewqav	62 rewqq
13 rew	62 rewqdsagas	62 rewqav
21 rewq	62 rewqgsab	62 rewqdsagas
62 rewq	62 rewq	62 rewqgsab
38 dfgds	62 rewqq	62 rewq
100 abcdfdsa	62 rewqerw	62 fdsae
100 defea	65 ere	65 ere
100 fadfagqe	92 reqw	92 reqw
100 gdagdsagdsa	93 ewqreer	93 ewqreer
100ewrqewtwqbaf	100 fadfagqe	100 abc
62 rewqerw	100 gdagdsagdsa	100 def
1023 a9	100 ewrqewtwqbaf	100 fadfag
62 rewqq	100 ewrqewtwq	100 gdagdsa
34 a5	100 abcdfdsa	100 ewrqewtwq
62 rewqav	100 defea	100 abcdfdsa
3 a1	100 abc	100 defea

62 rewqdsagas	100 def	100 fadfagqe
62 rewqgsab	100 fadfag	100 gdagdsagdsa
62 rewq	100 gdagdsa	100 ewrqewtwqbaf
62 fdsae	1023 a9	1023 a9