Algorithm EE398000 hw01
106061146 陳兆廷

1. Introduction:

The analysis and comparisons of the four sorting algorithms: selection sort, insertion sort, bubble sort and shaker sort.

The goal of the algorithms is to sort a list of vocabulary into alphabetical order.

The input of the algorithms is a list of vocabulary, which is in a form of 2d array. The output of the algorithms is a list of sorted, alphabetically ordered vocabulary.

2. Approach:

a. Analysis:

Analysis of **Selection Sort**:

i. **Intro:** Find minimum value, put it in the front of array.

ii. **Proof:** Each iteration in the ith iterations, it brings the smallest value among the n-1 items to the front of the array. Therefore, the algorithm finds the values in ascending order, undoubtly.

iii. Space complexity:

Fixed part: int: (i, j, k, n), *char: (tmp)

Variable part: *char (A) x n

Total: n (*char) + 4

iv. Time complexity:

Statement	s/e	Freq.	Total
Algorithm SelectionSort(**A, n**)	0	0	0
{	0	0	0
for i := 1 **to** n **do** {	n+1	1	n+1
j := i;	1	n	n
for k := i+1 **to** n **do** {	n+1	n	n(n+1)
if (A[k] < A[j]) then j = k	2 or 1	n(n+1)	n(n)*2or1
}	0	0	0
tmp = list[i]; list[i] = list[j]; list[j] = tmp;	3	n	3n
}	0	0	0
}	0	0	0
Total			$3n^2$ +4n+0

*since aveg of i is n/2, I calculate T(P) with i = n/2.

Best case: $2n^2$ +4n+2 (O(n^2)).

Worst Case: $3n^2$ +4n+2 (O(n^2)).

Average Case: O(n^2).

Therefore, the time complexity of Selection sort is O(n^2).

Analysis of **Insertion Sort**:

i. **Intro:** Compare with the minimum value, if true, swap, put it in the front of array.

ii. **Proof:** Like Selection sort, but Insertion sort gradually inserts the items from 2 to n into array in the front, in the right order.

iii. Space complexity:

Fixed part: int: (i, j, n), *char: (tmp)

Variable part: *char (A) x n

Total: n (*char) + 3

iv. Time complexity:

Statement	s/e	Freq.	Total
Algorithm InsertionSort(**A, n**)	0	0	0
{	0	0	0
for j := 2 **to** n **do** {	n-1	1	n-2
tmp := A[j];	1	n-2	n-2
i := j - 1;	1	n-2	n-2
while ((i >= 1) **and** (tmp < A[i])) **do** {	n (*)	n-2	n^2-2n
A[i + 1] := A[i]; i := i-1;	2	(n-2)(c)n	(2n-4)cn
}	0	(n-2)(c)n	0
A[i + 1] = tmp;	1	n-2	n-2
}	0	0	0
}	0	0	0
Total			(1+2c)n^2

*Assume >=, <, and are all performed. (I recall that if i<1, computer won't bother do the rest computation.)

Best case: if c = 0 or 1, it is mostly sorted, it won't do anything. (O(1))

Worst case: if c ~= n, it is almost decreasing, it is ~ (1+2c)n^2.(O(n^2))

Average case: O(n^2) since most of the case require sorting.

Therefore, the time complexity of Selection sort is O(n^2).

Analysis of **Bubble Sort**:

i. **Intro:** Swap whenever it's in the wrong order.

ii. **Proof:** It swaps whenever two neighbors are not in the right order. From ith iteration in 1 to n-1 iterations, it does the aforementioned behavior from n to i+1. This way, no single thing is in the wrong order.

iii. Space complexity:

Fixed part: int: (i, j, n), *char: (tmp)

Variable part: *char (A) x n

Total: n (*char) + 3

iv. Time complexity:

Statement	s/e	Freq.	Total

Algorithm BubbleSort(A, n)	0	0	0
{	0	0	0
for i := 1 **to** n - 1 **do** {	n	1	n
for j := n **to** i + 1 **step** -1 **do** {	n	n-1	n^2-n
if (A[j] < A[j – 1]) {	1	n-1	n-1
tmp = A[j]; A[j] = A[j-1]; A[j-1] = tmp;	3	c(n-1)	3c(n-1)
}	0	0	0
}	0	0	0
}	0	0	0
}	0	0	0
Total			n^2+…

Best case: if c = 0 or 1, the 2nd iteration does nothing, then it is (O(n)).

Worst case: if c ~= n, it is almost decreasing, it is ~ n^2+…(O(n^2))

Average case: O(n^2) since most of the case require sorting.

Therefore, the time complexity of Bubble sort is O(n^2).

Analysis of **Shaker Sort**:

i. **Intro:** Swap whenever it's in the wrong order but goes from both ends repeatedly.

ii. **Proof:** Like Bubble sort, but it reduces the 1 to n cycle to half of it, and repeat the swapping from n to i+1 again, from i to n-1. Same as Bubble sort, no single thing is in the wrong order.

iii. Space complexity:

Fixed part: int: (j, l, r, n), *char: (tmp)

Variable part: *char (A) x n

Total: n (*char) + 4

iv. Time complexity:

Statement	s/e	Freq.	Total
Algorithm ShakerSort(A, n)	0	0	0
{	0	0	0
l := 1; r := n;	2	1	2
while l <= r **do** {	n+1	1	n+1
for j := r **to** l + 1 **step** -1 **do** {	n	n	n^2
if (A[j] < A[j – 1]) {	c	n	cn
tmp = A[j]; A[j] = A[j-1]; A[j-1] = tmp;	3	n	3n
}	0	n	0
}	0	n	0
l = l + 1;	1	n	1n
for j := l **to** r - 1 **do** {	n	n	n^2
if (A[j] > A[j + 1]) {	c	n	cn
tmp = A[j]; A[j] = A[j+1]; A[j+1] = tmp;	3	n	3n
}	0	n	0
}	0	n	0
r := r – 1;	1	n	1n
}	0	0	0
}	0	0	0
Total			2n^2…

Best case: if c = 0 or 1, 2nd iterations do nothing, then it is (O(n)).

Worst case: if c ~= 1/2n, it is almost decreasing, it is(O(n^2)).

Average case: O(n^2) since most of the case require sorting, but it goes 2 times slower than bubble sort.

Therefore, the time complexity of Shaker sort is O(n^2).

Comparing Table:

	Selection	Insertion	Bubble	Shaker
Space	n (*char) + 4	n (*char) + 3	n (*char) + 3	n (*char) + 4
Time	**O(n^2)**	**O(n^2)**	**O(n^2)**	**O(n^2)**

Prediction: Shaker > Bubble ? Insertion > Selection

The reason of Insertion > Selection is because the comparing step is lesser in Insertion sort. Shaker sort reduces the swapping steps by comparing from 2 ends.

The speed of the algorithms depends on whether writing data(swapping) or going through more steps is faster. Based on my learning in Computer Architecture, writing data is way slower.

Therefore, **my prediction is: Insertion > Selection > Shaker > Bubble**.

b. Implementation: (hw01.c on NTHUEE workstation, gcc 4.1.2)

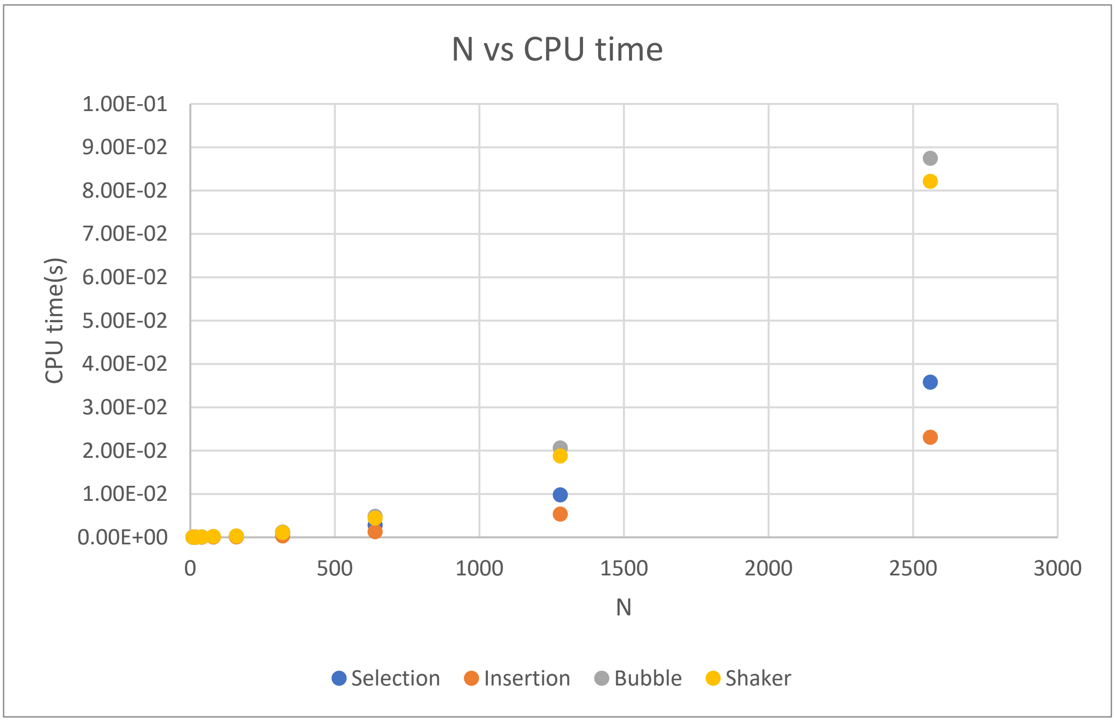
3. Result:

a. Result Table:

N	Selection	Insertion	Bubble	Shaker
10	9.90E-07	7.30E-07	1.11E-06	1.12E-06
20	6.79E-06	5.09E-06	1.00E-05	9.99E-06
40	2.33E-05	1.44E-05	3.96E-05	4.04E-05
80	8.60E-05	5.27E-05	1.64E-04	1.64E-04
160	1.33E-04	7.21E-05	2.77E-04	2.65E-04
320	1.14E-03	3.03E-04	1.13E-03	1.07E-03
640	2.76E-03	1.24E-03	4.81E-03	4.39E-03
1280	9.75E-03	5.32E-03	2.06E-02	1.87E-02
2560	3.58E-02	2.31E-02	8.74E-02	8.21E-02

b. Observation:

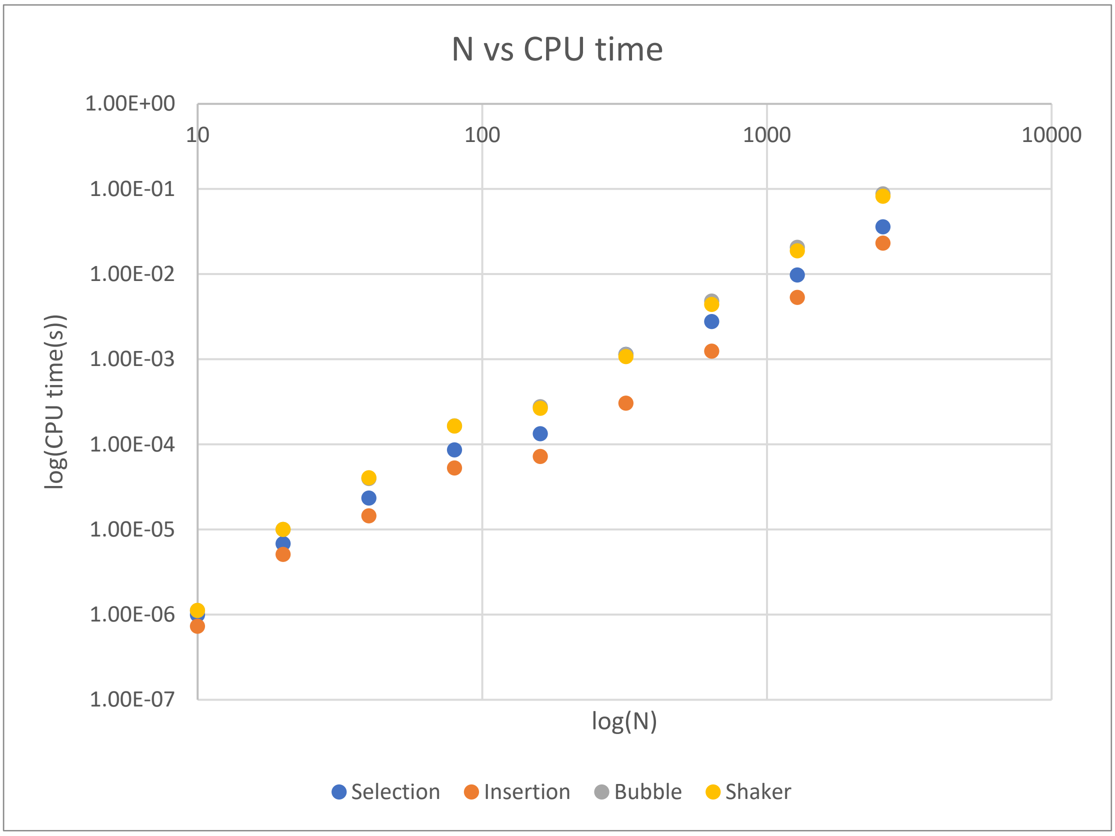
- N vs CPU time chart:



Speed: Insertion > Selection > Shaker > Bubble.

It seems that the result meets by prediction. Furthermore, the time complexities of the four algorithms are O(n^2), same as those that I've calculated.

- Log(N) vs log(CPU time) chart:



When y-axis is logged, the graph looks linear. It indicates that they are O(n^2).

Logged x-axis is just for good spacing between different Ns.

*The N=10 & N=40 case's bubble sort is faster than shaker sort. They might just be the special case when N is small.

c. Conclusion:
   i. Experimented Speed: **Insertion > Selection > Shaker > Bubble**.
   ii. Time Complexity: **Insertion = Selection = Shaker = Bubble = O(n^2)**.
   iii. Best Case: **Insertion > Shaker = Bubble > Selection.**
   iv. Space Complexity: **Insertion = Selection = Shaker = Bubble = theta(1)**
   v. **Swapping (data writing) is much slower than comparing.**