CO322 Data Structures and Algorithms
Lab04 : Simple sorting algorithms

E/14/158
Gihan Chanaka Jayatilaka

20-04-2018

Contents

1 Overview

This report analyzes the following sorting algorithms.

- Bubble sort
- Selection sort
- Insertion sort

These algorithms were implemented on java and tested on integer **(int)** arrays of different sizes. The arrays were populated with random integers generated by java Random class with bounds of [0, twice the array size).

"Which algorithm is better?" depends on several factors such as,

- Accuracy of the algorithm
- Code complexity of the algorithm
- Efficiency of the algorithm

These factors can be evaluated both theoretically and experimentally.

2 Algorithm implementation

2.1 Bubble sort

```java
static void bubble_sort(int [] data) {
    for(int iter=0;iter<data.length;iter++){
        boolean sorted=true;
        for(int x=0;x<data.length-1-iter;x++){
            if(data[x]>data[x+1]){
                sorted=false;
                swap(data,x,x+1);
            }
        }
        if(sorted)break;
    }
}
```

2.2 Selection sort

```java
static void selection_sort(int [] data) {
    for(int i=0;i<data.length-1;i++){
        int minIndex=i;
        for(int j=i+1;j<data.length;j++){
            if(data[minIndex]>data[j]){
                minIndex=j;
            }
        }
        swap(data,i,minIndex);
    }
}
```

2.3 Insertion sort

```
static void insertion_sort(int [] data) {
    for(int i=1;i<data.length;i++){
        for(int j=i-1;true;j--){
            if(j==-1|| data[i]>data[j]){
                //circular swap operation
                int temp=data[i];
                for(int ii=i;ii>j+1;ii--)data[ii]=data[ii-1];
                data[j+1]=temp;
                break;
            }
        }
    }
}
```

3 Analysis factors

3.1 Accuracy of the algorithm

The accuracy of all the algorithms considered in this exercise is 100%. Therefor it is not a suitable comparison metric to choose the "better algorithm" in this case.

3.2 Code complexity of the algorithms

All three algorithms were implemented using **nested loops, comparison and swap** operations. The insertion sort has **circular swap** which is relatively complicated than the other algorithms.
The code is fairly simple for all three algorithms.

3.3 Efficiency of algorithms

Efficiency of an algorithm is the inverse of the resource consumption for the algorithm to run. The resources consumed are

- Computation time – Time complexity
- Memory – Space complexity

Both these complexities could be analyzed theoretically or experimentally. In this report, the theoretical analysis is done for worst case scenarios **(Big Oh)**. The experimental analysis is done for *average case* by generating the unsorted arrays by a uniform random distribution of integers.

4 Theoretical analysis

4.1 No of operations

Algorithm	Best case		Worst case	
	No. of comparisions	No. of swaps	No. of comparisions	No. of swaps
Bubble sort	N-1	0	$\frac{N(N-1)}{2}$	$\frac{N(N-1)}{2}$
Selection sort	N-1	0	$\frac{N(N-1)}{2}$	$\frac{N(N-1)}{2}$
Insertion sort	$\frac{N(N-1)}{2}$	0	$\frac{N(N-1)}{2}$	$\frac{N(N-1)}{2}$

4.2 Big "O"

Note:
The space complexities are calculated by the additional space needed for the sort function. Since the unsorted array is passed by value in to the function, the space required for the array is neglected.

Algorithm	Time complexity	Space complexity
Bubble sort	$O(N^2)$	$O(1)$
Selection sort	$O(N^2)$	$O(1)$
Insertion sort	$O(N^2)$	$O(1)$

5 Experimental

5.1 Procedure

All three algorithms were evaluated against integer arrays of sizes **100, 1000, 5000, 10,000, 50,000 and 100,000**. These arrays were filled by random numbers (from a uniform random distribution).
The 3 cases were chosen to be

- **Average case :** The randomly filled array.
- **Best case :** The sorted array.
- **Worst case :** The reversed array.

The time taken for an algorithm to sort (or identify the array as a sorted array in the best case scenarios) were measured by the System.currentTimeMillis() function in java.

5.2 Results

5.2.1 Tabulated results for different order of N

N	Bubble sort			Selection sort			Insertion sort		
	Average /(ms)	Best /(ms)	Worst /(ms)	Average /(ms)	Best /(ms)	Worst /(ms)	Average /(ms)	Best /(ms)	Worst /(ms)
100	0	0	1	0	0	0	0	0	0
500	4	0	2	3	1	1	1	0	2
1000	1	0	1	0	1	3	3	0	3
5000	27	0	27	9	8	17	5	0	9
10000	130	0	103	34	33	82	21	0	34
50000	5130	0	2582	829	830	2192	430	0	856
100000	19837	1	10290	3328	3310	8783	1749	0	3608

5.2.2 Results for different N in same order

N	Bubble sort			Selection sort			Insertion sort		
	Average /(ms)	Best /(ms)	Worst /(ms)	Average /(ms)	Best /(ms)	Worst /(ms)	Average /(ms)	Best /(ms)	Worst /(ms)
10000	208	0	170	59	55	84	38	0	44
20000	756	0	410	136	131	360	69	0	139
30000	1831	0	924	299	297	806	152	0	305
40000	3224	0	1638	531	527	1461	269	0	543
50000	5069	0	2559	826	817	2275	423	0	848
60000	7385	0	3693	1187	1177	3252	610	1	1225
70000	10031	0	5045	1617	1603	4427	831	0	1675
80000	13116	0	6548	2113	2097	5856	1094	1	2235
90000	16657	0	8287	2673	2659	7374	1400	1	2885
100000	20648	0	10262	3327	3293	9201	1733	0	3608

Figure 01:Bubble sort execution times

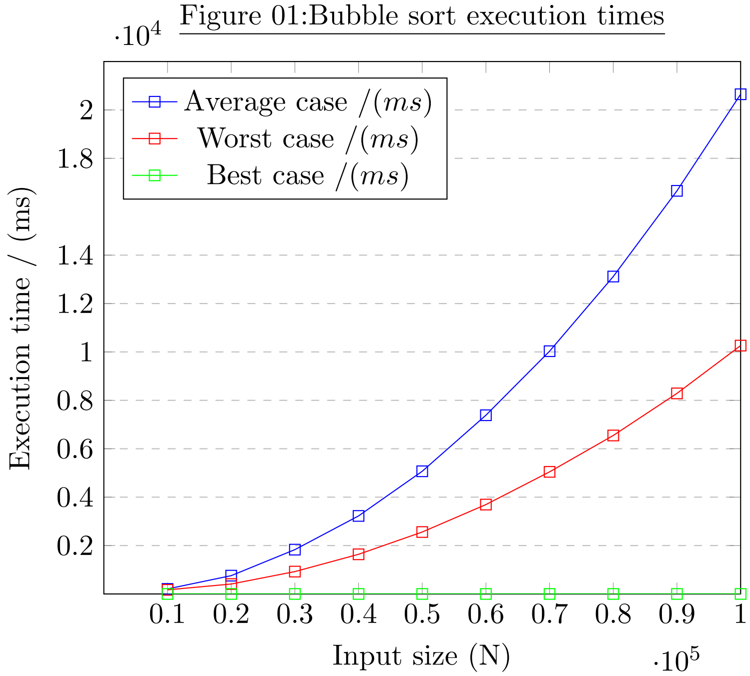


Figure 02:Selection sort execution times

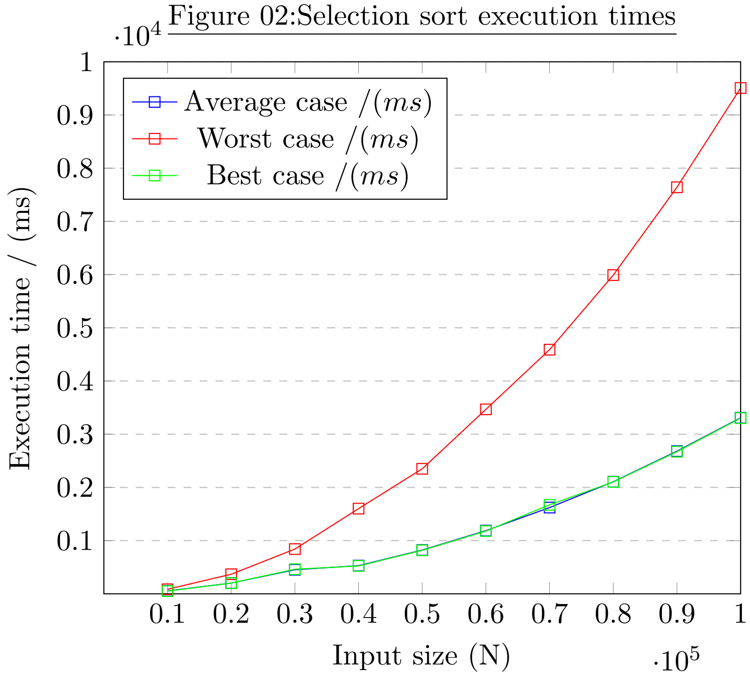


Figure 03:Insertion sort execution times

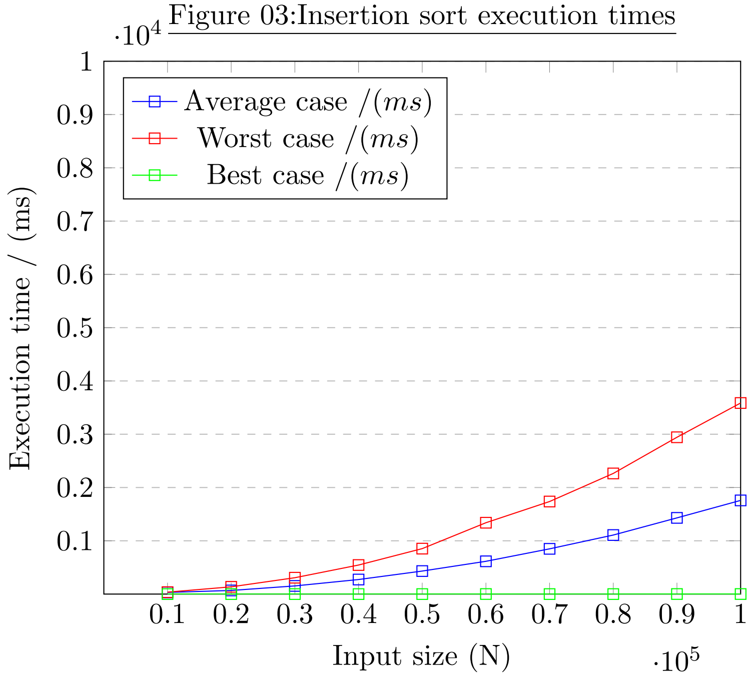
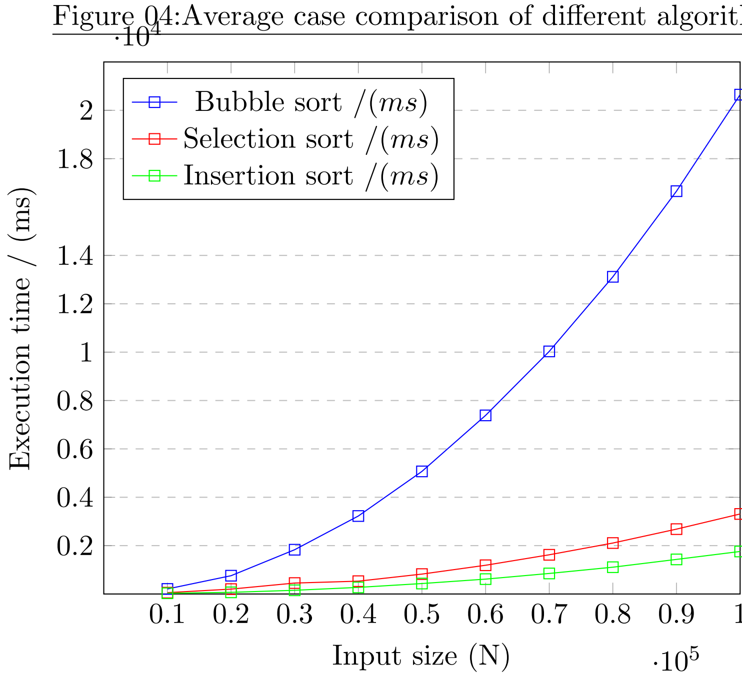


Figure 04:Average case comparison of different algorithms



5.3 Observations and conclusions

- All sorting algorithms identify a sorted array and terminates very quickly except for the insertion sort.
- The average time efficiency of the sorting algorithms increase in the order of **Bubble** $<$ **Selection** $<$ **Insertion**.
- The shape of $Time - N$ graphs match the theoretical prediction $O(N^2)$.
- Surprisingly the *worst case* is faster than the average case in Bubble sort.