Analysis and Conclusions of the First Laboratory

Basic Sorting Algorithms

Student: Thaís Steinmuller Farias
thais.farias@telecom-sudparis.eu
Undergraduate at
Telecom SudParis

Évry-Courcouronnes, October 2022

1 Introduction and Methods

The following laboratory had as its main objective to code and make an analysis of three of the most common sorting algorithms, as seen during classes: bubble sort, insertion sort and selection sort. It was coded using the C programming language and the Unix environment, as seen during classes [Kernighan e Ritchie 1988]. As discussed in class, bubble sort works by seeing a set of adjacent positions of the array and switching them based on their value, insertion sort works by moving the elements of the array one by one to their right position and selection sort functions by looking for the smallest element of the array and bringing it upfront, then dividing the array into sorted part and unsorted part [Ganapathi e Chowdhury 2021].

With that in mind, the code analysis was divided in two cases, them being:

- Case 1: Sorting algorithms applied to a random generated array.
- Case 2: Sorting algorithms applied to a reverse ordered array.

Those two cases were applied 5 times with different sized arrays of length L, L being the range 10, 100, 1000, 10000, 100000. The time was then measured for each case, as seen in Fig. 1.

After that, three samples of data were collected and put into tables in order to make a graph and make an analysis of each algorithm's behavior (Fig. 2). With that data, the media of the values of each trial was taken and transformed into two graphs, one referring to the random generated array trials (Fig. 3) and the other one to the descending ordered array (Fig. 4).

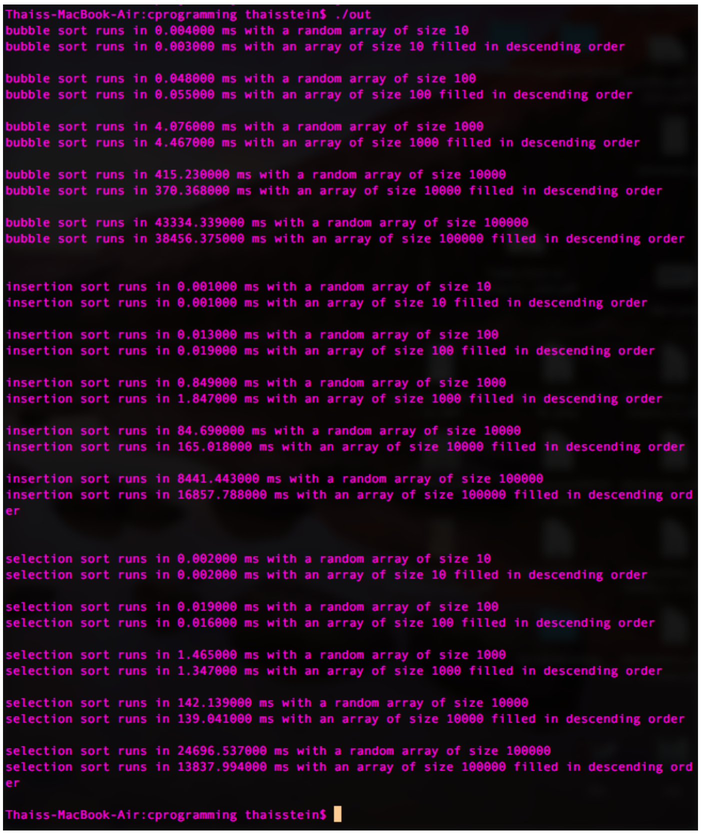
2 Data Analysis

As we can see looking into the first case graph (Fig. 3), the bubble sort is a lot slower than the the other two algorithms. I could easily realize that because of the time it took for my terminal to execute the bubble sort. That happens because there's a lot of operations needed to finish the sorting when compared to the others, so its slope in the graph is the steepest.

In that same case, comparing the insertion and the selection sort it's clear in the graph that insertion sort has a better performance. That means that the insertion sort algorithm makes less comparisons that the other ones in the average case of a random generated list.

Inspecting the second case, with the reverse ordered array, we can see that the efficiency of the insertion sort drops over time. That makes sense observing how the algorithm works, once it has to do way more operations by comparing and swapping all of this array's elements one by one, being its worst case. So, despite of having a great efficiency in case 1, insertion sort ended up being worse than both bubble and selection sort.

The bubble sort's efficiency stays almost the same when compared to the first case, with a slightly steeper slope due to the increase of needed operations when the vector is ordered in reverse. Therefore bubble sort is the slowest algorithm when compared to the other two in both cases.



```
Thaiss-MacBook-Air:cprogramming thaisstein$ ./out
bubble sort runs in 0.004000 ms with a random array of size 10
bubble sort runs in 0.003000 ms with an array of size 10 filled in descending order

bubble sort runs in 0.048000 ms with a random array of size 100
bubble sort runs in 0.055000 ms with an array of size 100 filled in descending order

bubble sort runs in 4.076000 ms with a random array of size 1000
bubble sort runs in 4.467000 ms with an array of size 1000 filled in descending order

bubble sort runs in 415.230000 ms with a random array of size 10000
bubble sort runs in 370.368000 ms with an array of size 10000 filled in descending order

bubble sort runs in 43334.339000 ms with a random array of size 100000
bubble sort runs in 38456.375000 ms with an array of size 100000 filled in descending order


insertion sort runs in 0.001000 ms with a random array of size 10
insertion sort runs in 0.001000 ms with an array of size 10 filled in descending order

insertion sort runs in 0.013000 ms with a random array of size 100
insertion sort runs in 0.019000 ms with an array of size 100 filled in descending order

insertion sort runs in 0.849000 ms with a random array of size 1000
insertion sort runs in 1.847000 ms with an array of size 1000 filled in descending order

insertion sort runs in 84.690000 ms with a random array of size 10000
insertion sort runs in 165.018000 ms with an array of size 10000 filled in descending order

insertion sort runs in 8441.443000 ms with a random array of size 100000
insertion sort runs in 16857.788000 ms with an array of size 100000 filled in descending ord
er


selection sort runs in 0.002000 ms with a random array of size 10
selection sort runs in 0.002000 ms with an array of size 10 filled in descending order

selection sort runs in 0.019000 ms with a random array of size 100
selection sort runs in 0.016000 ms with an array of size 100 filled in descending order

selection sort runs in 1.465000 ms with a random array of size 1000
selection sort runs in 1.347000 ms with an array of size 1000 filled in descending order

selection sort runs in 142.139000 ms with a random array of size 10000
selection sort runs in 139.041000 ms with an array of size 10000 filled in descending order

selection sort runs in 24696.537000 ms with a random array of size 100000
selection sort runs in 13837.994000 ms with an array of size 100000 filled in descending ord
er

Thaiss-MacBook-Air:cprogramming thaisstein$
```

Fig. 1: Print screen of one of the code output in the terminal, containing each of the cases for the sorting algorithms. Reference: Personal archive

.

First Trial			Second Trial			Third Trial		
Bubble Sort	Random Array	Descending Order	Bubble Sort	Random Array	Descending Order	Bubble Sort	Random Array	Descending Order
10	0	0,004	10	0	0,003	10	0	0,003
100	0,075	0,134	100	0,048	0,055	100	0,073	0,087
1000	5,808	6,679	1000	4,076	4,467	1000	5,391	3,53
10000	388,019	375,815	10000	415,23	370,368	10000	410,014	505,198
100000	43885,853	36351,324	100000	43334,375	38456,375	100000	42173,237	36426,852
Insertion Sort	Random Array	Descending Order	Insertion Sort	Random Array	Descending Order	Insertion Sort	Random Array	Descending Order
10	0,002	0,002	10	0,001	0,001	10	0,002	0,002
100	0,012	0,032	100	0,013	0,019	100	0,017	0,029
1000	0,826	1,589	1000	0,849	1,847	1000	1,072	1,577
10000	79,323	157,829	10000	84,69	165,018	10000	93,832	207,068
100000	8292,775	16639,005	100000	8441,443	16857,788	100000	8150,963	17069,01
Selection Sort	Random Array	Descending Order	Selection Sort	Random Array	Descending Order	Selection Sort	Random Array	Descending Order
10	0,003	0,001	10	0,002	0,002	10	0,001	0,002
100	0,02	0,017	100	0,019	0,016	100	0,017	0,016
1000	1,533	1,682	1000	1,456	1,347	1000	1,362	1,354
10000	194,001	147,349	10000	142,139	139,041	10000	146,455	134,306
100000	25678,878	15546,462	100000	24696,537	13837,994	100000	15399,681	13755,041

Fig. 2: Table containing the data collected from the code output on 3 different trials. Reference: Personal archive

.

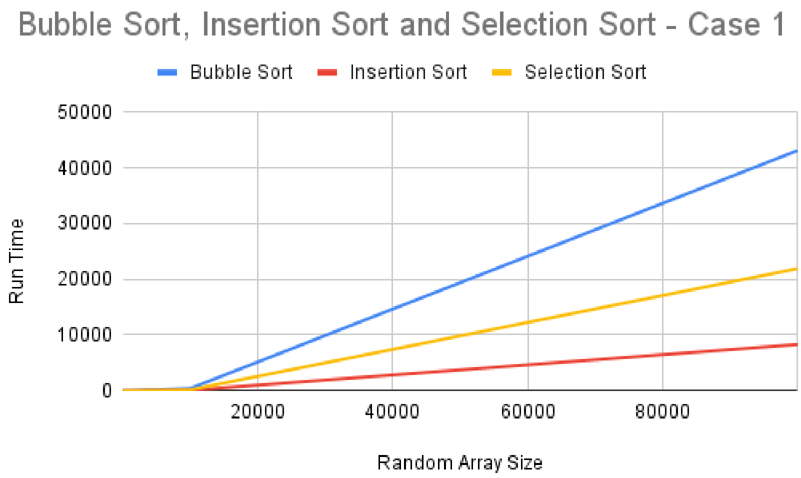


Fig. 3: Graph that represents the run time of the three sorting algorithms versus the size of the random sorted array. Reference: Personal archive

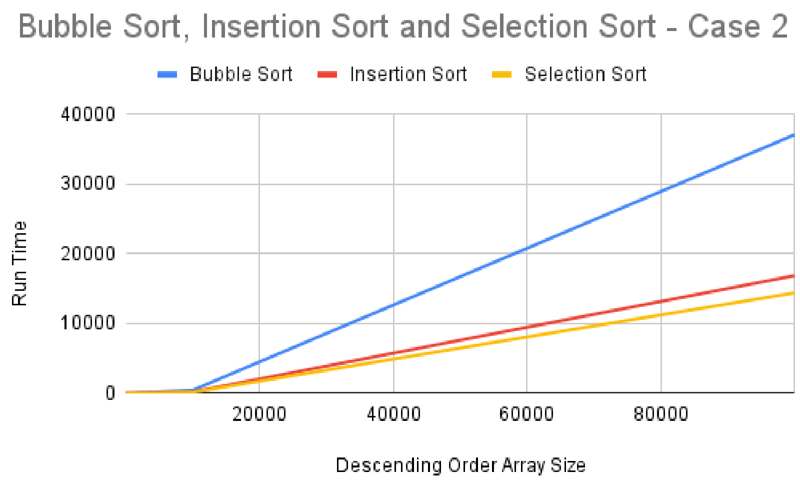


Fig. 4: Graph that represents the run time of the three sorting algorithms versus the size of the descending ordered array. Reference: Personal archive

References

GANAPATHI, P.; CHOWDHURY, R. Parallel divide-and-conquer algorithms for bubble sort, selection sort and insertion sort. *The Computer Journal*, 2021.

KERNIGHAN, B. W.; RITCHIE, D. M. *The C programming language*. [S.l.]: Pearson Educación, 1988.