Eduard Forés Ferrer

ORDER ALGORITHMS SPEED IN DIFFERENTS LANGUAGES

RESEARCH AND ENTREPRENEURSHIP IN SECURITY AND ARTIFICIAL INTELLIGENCE

ALEJANDRO ARENAS MORENO

UNIVERSITY MASTER'S DEGREE IN COMPUTER SECURITY ENGINEERING AND ARTIFICIAL INTELLIGENCE

UNIVERSITAT ROVIRA i VIRGILI

Tarragona
2021

Abstract

The main purpose of this paper is compare the speed of the different sorting algorithms in different languages. First of all, we will explain what kind of sorting algorithms there are and how it works. Then, we will execute different algorithms in Python and java with different sizes of data and we will compare the results between two languages.

As we know by the literature related with this topic not all algorithms have the same execution time. So, we won't compare the different algorithm between them otherwise we will compare different executions of every one and the results between the different executions of Python and Java. It is important to note that all algorithms work with the same data.

In this paper we will find a lot of graphics expressing the results, comparing the time results of Python and Java. As you will see in the "Results" part the faster algorithm to order data is Java with enough difference.

The results express the supremacy of the Java over Python because the timings are so much higher in Python than Java. At least for this task the Java in the matter of time is much more efficient.

1. Introduction

In computation and mathematics, an ordering algorithm is an algorithm that puts elements of a list or a vector in a sequence given by an order relation, that is, the output result must be a permutation - or rearrangement - of the input that satisfy the given order relation. The most used order relationships are numerical order and lexicographic order.

Efficient sorts are important for optimizing the use of other algorithms (such as search and merge) that require ordered lists for fast execution. It is also useful for putting data in canonical form and for generating human-readable results.

There are a lot of algorithms to order arrays and they are classified between two groups: stable or instable. In this paper we will work with the first group because they are the most popular and extended group around the world.

Before, we see the experiment and the results the paper explain some information about every algorithm used in this investigation.

The final objective of this work is know what is the most efficient language, between python and java, to process and order data. Finally, the hypothesis of the paper is “Could the sorting algorithms be faster in python than in java?”.

2. Order Algorithms

In the order algorithms world, as you read in the introduction, there are two groups to classify the algorithms, stable or instable algorithms. In this paper we will research the stable groups. In this part, you will receive information about every algorithm used in this investigation to know how it works and some qualities of each algorithm.

2.1. Bubble Sort

Bubble Sort is a simple sorting algorithm. It works by checking each item in the list to be sorted with the next, swapping them if they are in the wrong order. The entire list needs to be reviewed several times until no more exchanges are needed, which means that the list is sorted. This algorithm gets its name from the way items move up through the list during exchanges, like little "bubbles." It is also known as the direct exchange method. Since it only uses comparisons to operate elements, it is considered a comparison algorithm, being one of the simplest to implement. [2]

2.2. Cocktail Sort

The bidirectional bubble sort or cocktail is a sort algorithm that arises as an improvement of the bubble sort algorithm.

The way to work with this algorithm is to order at the same time by both ends of the vector. So after the first iteration, both the smallest and largest elements will be in their final positions.

We make an ascending journey (from the first element to the last), we take the first element and compare it with the next one, if the next one is lower, we move it to the previous position, in this way at the end of the list we have the largest one. Once the ascending series is finished, we make a descending journey (from the last element to

the first) but this time we are left with the minors to which we advance positions instead of delaying them as we did in the ascending series. We repeat the series alternately but reducing the scope at its ends since we will already have the lowest and highest values in the list there, until there are no elements left in the series; In the example pseudocode: Up to (left> right). [3]

2.3. Insertion Sort

Insertion sort is a very natural way for a human to sort, and can easily be used to sort a deck of arbitrarily numbered cards. It requires $O(n^2)$ operations to sort a list of n items.

Initially you have only one element, which is obviously an ordered set. Then, when there are k elements ordered from lowest to highest, the element $k + 1$ is taken and compared with all the elements already ordered, stopping when a minor element is found (all major elements have been shifted one position to the right) or when elements are no longer found (all elements were displaced and this is the smallest). At this point the element $k + 1$ is inserted and the other elements must be moved. [4]

2.4. Bucket Sort

bucket sort or bin sort is an ordering algorithm that distributes all the items to be sorted among a finite number of boxes. Each box can only contain items that meet certain conditions. In the example, those conditions are ranges of numbers. The conditions must be mutually exclusive, to avoid that an element can be classified in two different boxes. Then each of these boxes is ordered individually with another sort algorithm (which could be different depending on the box), or this algorithm is applied recursively to obtain boxes with fewer elements. This is a generalization of the Pigeonhole sort algorithm. When the elements to be ordered are uniformly distributed, the computational complexity of this algorithm is $O(n)$. [5]

2.5. Counting Sort

The ordering by accounts (counting sort in English) is an ordering algorithm in which the number of elements of each class is counted and then ordered. It can only be used therefore to order elements that are countable (such as whole numbers in a certain interval, but not real numbers, for example).

The first step is to find out what is the interval within which the data to be sorted is (minimum and maximum values). Then a vector of integers is created with as many elements as there are values in the interval [minimum, maximum], and each element is given the value 0 (0 occurrences). After this, all the elements to be ordered are traversed and the number of occurrences of each element is counted (using the vector that we have created). Finally, it is enough to traverse this vector to have all the elements ordered. [6]

2.6. Merge Sort

The merge sort algorithm is a stable external sort algorithm based on the divide and conquer technique that was invented by John von Neumann in 1945. It is of complexity $O\ (n\ \log n)$. [7]

Conceptually, sorting by merge works as follows:

1. If the length of the list is 0 or 1, then it is already sorted. In another case:
2. Divide the unordered list into two sublists about half the size.
3. Sort each sublist recursively applying the sort by mix.
4. Mix the two sublists into a single ordered list.

Merge sort incorporates two main ideas to improve your runtime:

1. A small list will take fewer steps to sort than a large list.
2. It takes fewer steps to build an ordered list from two ordered lists than from two unordered lists. For example, each list only needs to be interleaved once they are sorted.

2.7. Radix Sort

In computing, the Radix ordering is a sorting algorithm that orders integers by processing their digits individually. Because integers can represent character strings (for example, names or dates) and especially specially formatted floating-point numbers, radix sort is not limited to just integers. [8]

3. Experiment

To test these algorithms, we use the same one laptop MSI GF75 Thin 9SC-277XES. The tests consist in 5 tests for each algorithm where each test will have 3 runs with the same weight of data. For example, the first test will be with one array with 10 field of length and this test will run 3 times and the second test will be with 50 fields of length and it will be run 3 times more. The final time for each test will be the arithmetic median of the tree runs for each test.

One more important thing is that the java and python will work with the same lists to do all tests.

First of all, we will investigate different implementations to make the code the most similar possible to do the test more equitable. Then, we will create a small algorithm to create the different data files with the numbers to order. These files will be named "`test`"+`numberOfLength`+"`.txt`", for example, test10.txt. After it, we have the information to make the code in these two languages and the data to test the algorithms. Now is time to coding the algorithms in Java and Python. You can find these algorithms with all data test, the code to generate the data and the results of the experiment in my GitHub[1].

When, I finished the development and see the results of the timings I cannot believe the time difference between these 2 languages.

4. Results

In the next tables you can see the results for the different data length for Java and Python depending the algorithm. The numbers expressed in the tables are in microseconds and they are the arithmetic median of the three runs.

Bubble Sort					
	10	100	1000	10000	100000
JAVA	1,3667	58,0667	1871,4	99468,3	1,20E+07
PYTHON	0	826,9151	5538,4506	5679222,35	6,06E+08

Cocktail Sort					
	10	100	1000	10000	100000
JAVA	1,8333	44,9666	1981,9667	74989,3	9,16E+06
PYTHON	0	656,8432	52742,401	4973874,33	5,29E+08

Insertion Sort					
	10	100	1000	10000	100000
JAVA	1,2667	14,3	865,2	23488,6667	1,11E+06
PYTHON	0	0	28787,2155	2785735,21	2,96E+08

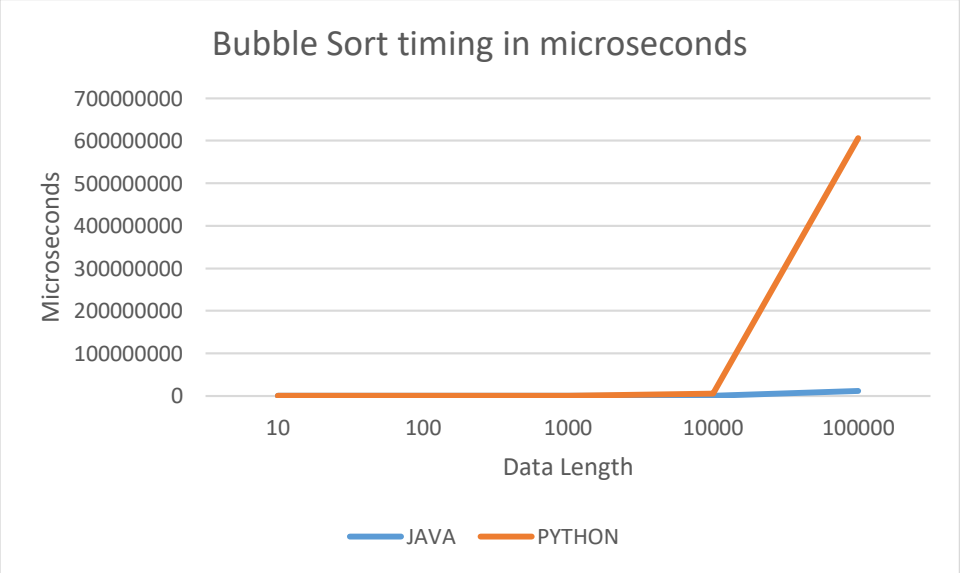
Bucket Sort					
	10	100	1000	10000	100000
JAVA	525,9	315,6	263,8333	356,6667	1,44E+03
PYTHON	0	487,6455	2170,7217	337286,234	4,21E+07

Counting Sort					
	10	100	1000	10000	100000
JAVA	1126,0667	650,5333	457,9	790,5666	1,67E+03
PYTHON	0	661,2142	991,4239	7306,81419	5,80E+04

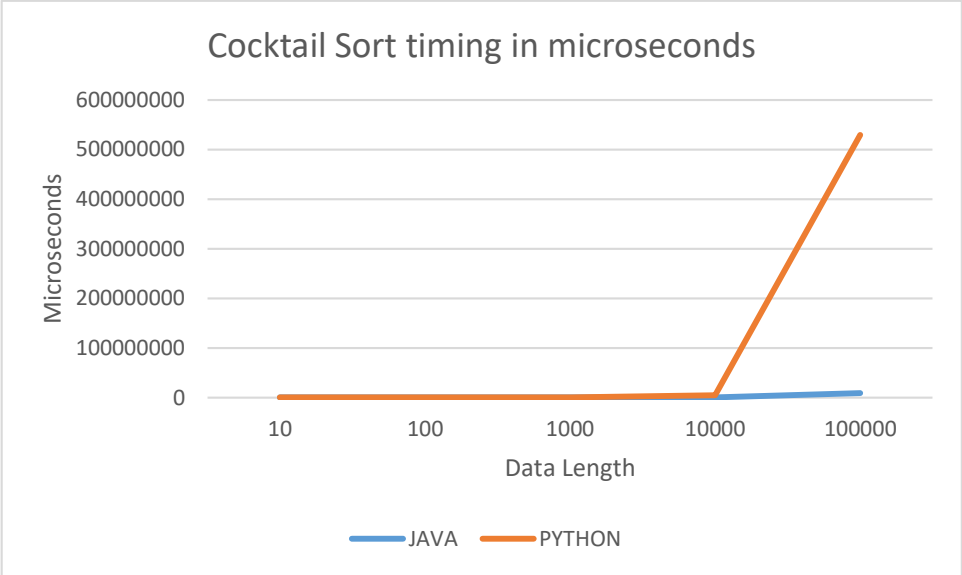
Merge Sort					
	10	100	1000	10000	100000
JAVA	4,6	30,5667	305,3667	1432,3667	1,60E+04
PYTHON	0	330,3686	2810,4782	33574,4222	3,70E+05

Radix Sort					
	10	100	1000	10000	100000
JAVA	19,3667	21,7333	298,333	1630,3667	7,38E+03
PYTHON	4462,4011	35567,2042	337124,825	352557,111	3,42E+07

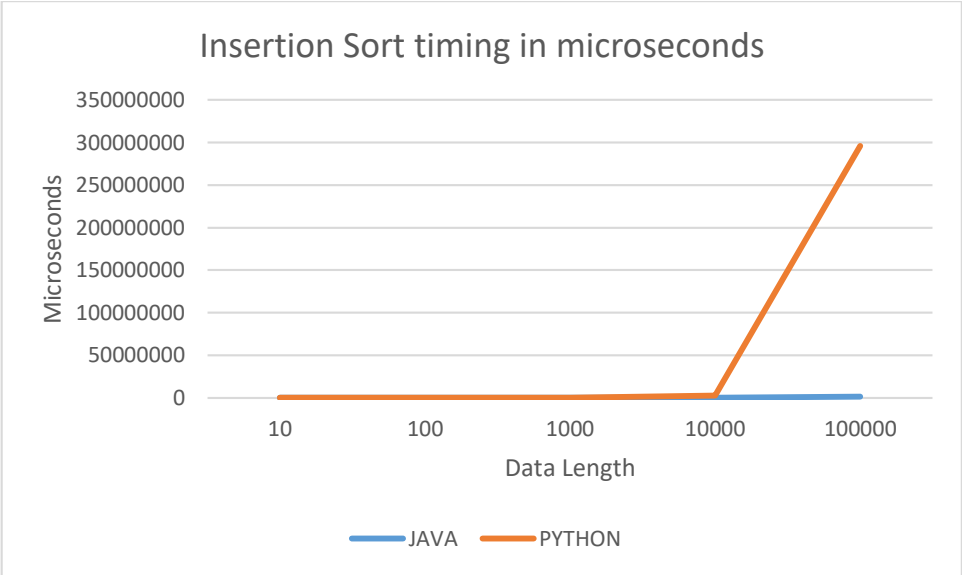
If we observe the time results in these tables we can see that python is much slower than Java ordering the files created for the experiment. When, I tried create some graphics to present in the paper the line of java is a straight line and the python's line is fired upwards braking the graph. Here you have the graphs:



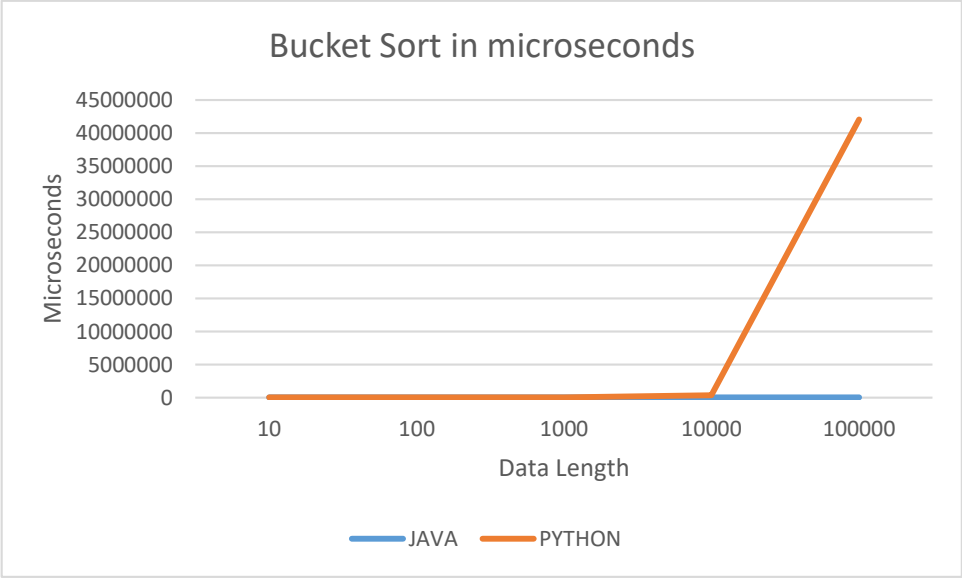
Graphic1. Bubble Sort timing line graph



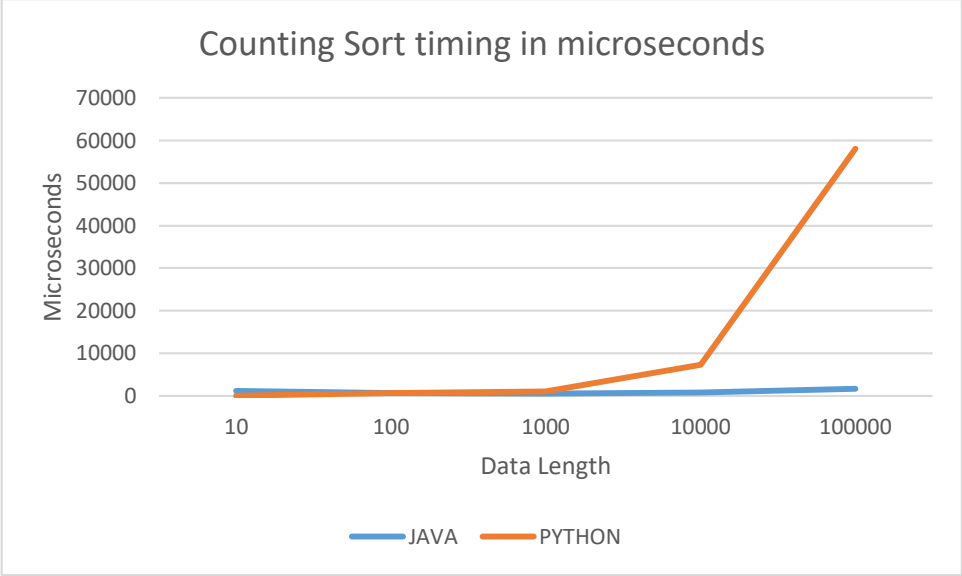
Graphic2. Cocktail Sort timing line graph



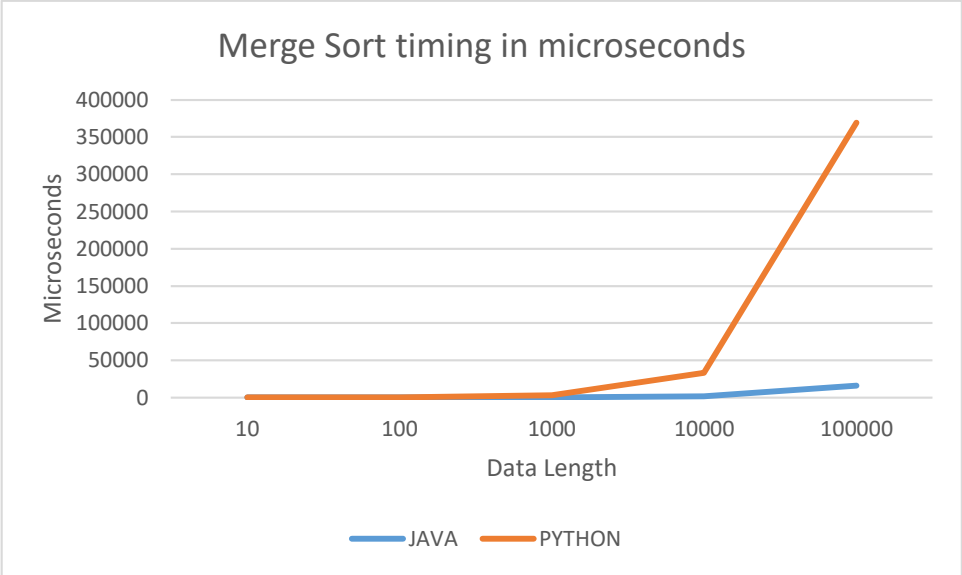
Graphic3. Insertion Sort timing line graph



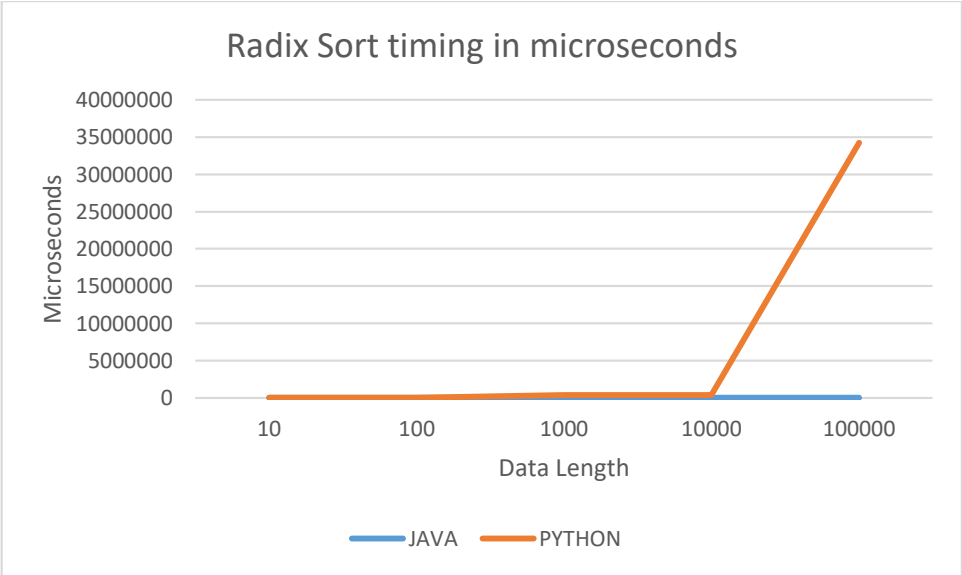
Graphic4. Bucket Sort timing line graph



Graphic5. Counting Sort timing line graph

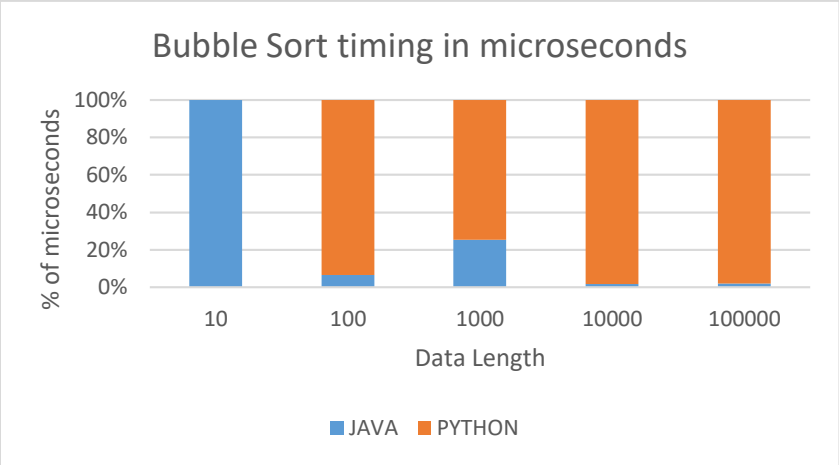


Graphic6. Merge Sort timing line graph

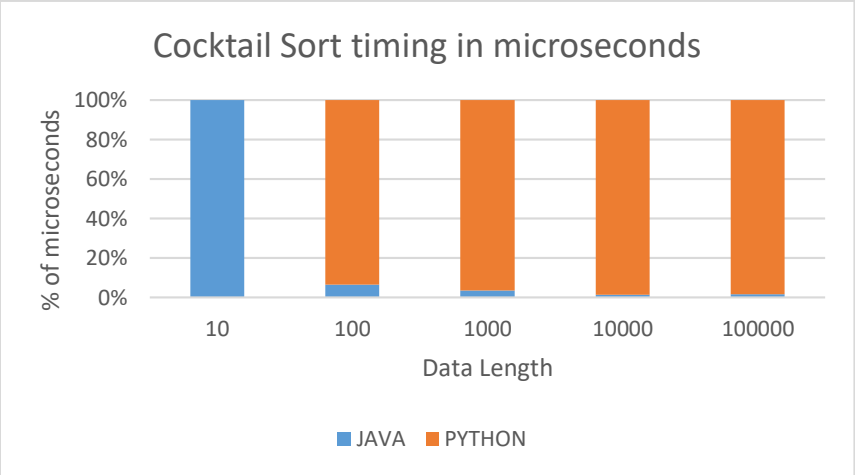


Graphic7. Radix Sort timing line graph

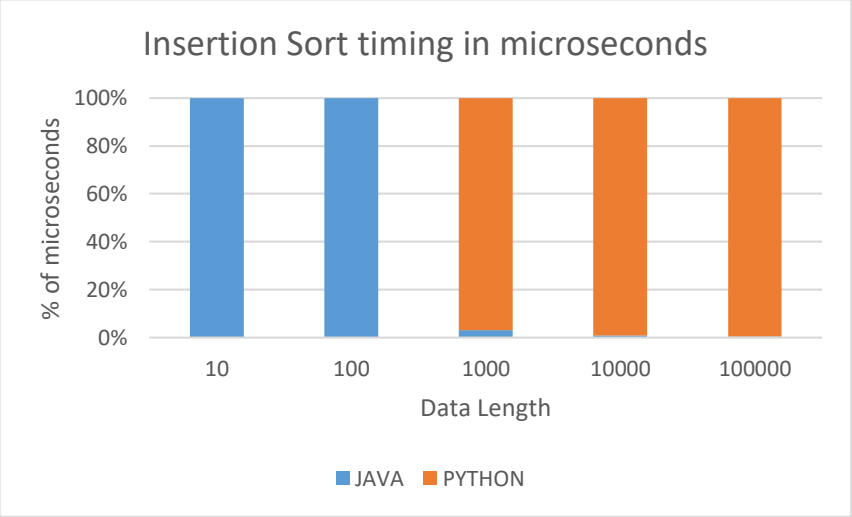
The merge and counting sort are the only ones that we can appreciate a little the movement of the Java line but the other graphics the Java line is almost nil. These graphics do not give us the necessary visual information to know what happen. Therefore, we can see these following graphics to appreciate the influence time of python respect the java time since these lasts graphics do not show us the information clearly.



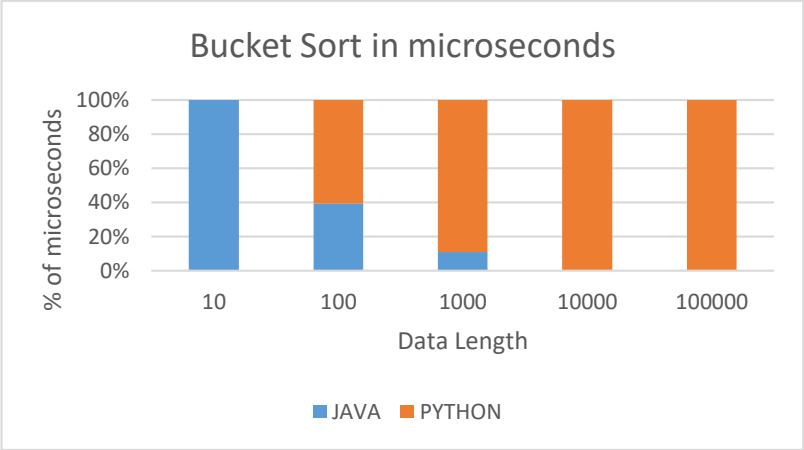
Graphic8. Bubble Sort 100% stacked column chart



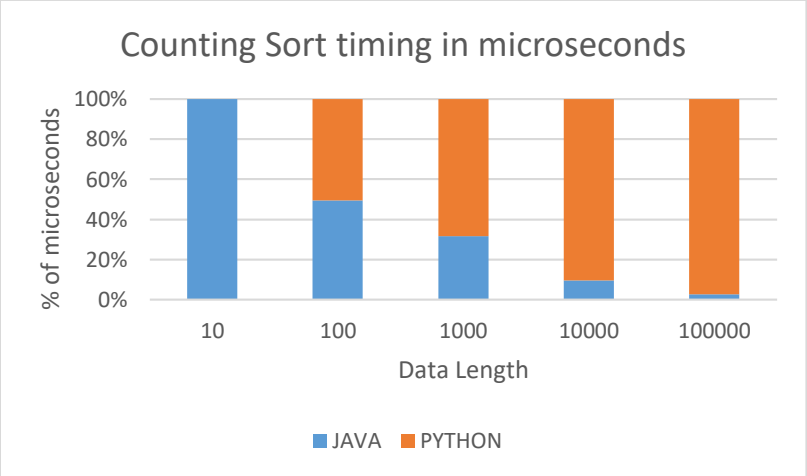
Graphic9. Cocktail Sort 100% stacked column chart



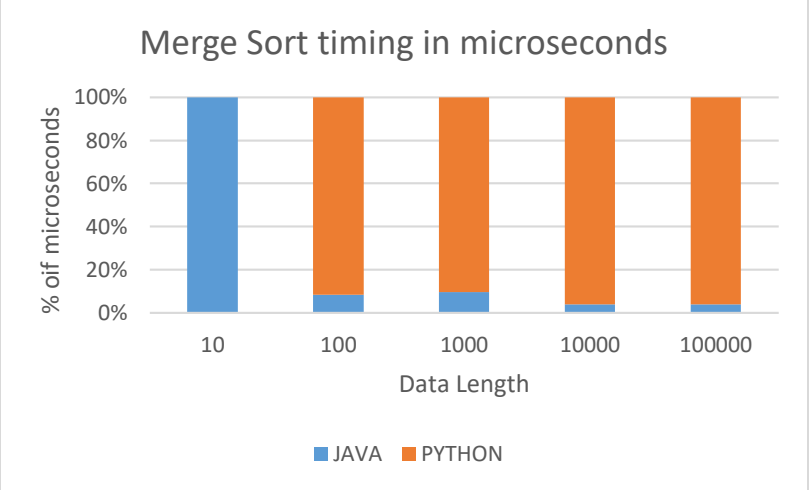
Graphic10. Insertion Sort 100% stacked column chart



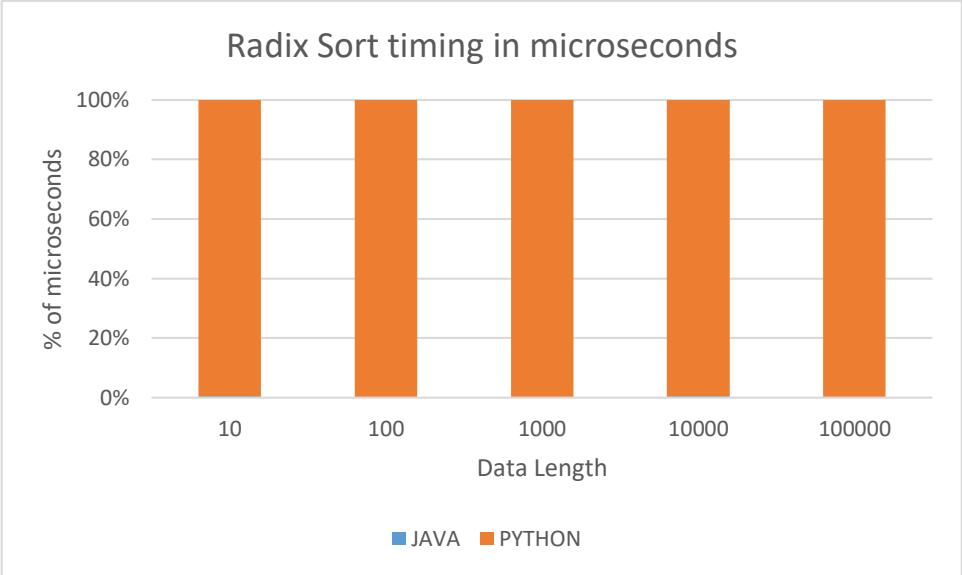
Graphic11. Bucket Sort 100% stacked column chart



Graphic12. Counting Sort 100% stacked column chart



Graphic13. Merge Sort 100% stacked column chart



Graphic11. Radix Sort 100% stacked column chart

With this graphics we can observe that the python when the data to order is higher than 100 units the timing regarding java is much more. However, if the data has very small length the python is faster than java. The problem is that in a real case the data length could be one million units and then python is so too slow to do some job to order the data.

5. Conclusion

In conclusion, my personal opinion is Java is very efficient with the timing than python and if you need a lot of velocity Java is a pretty good option but certainly python is not faster than java but the code of python is cleaner than Java and more elegant. So if you do not need a lot of speed and the code complexity is very high, python may be a good option anyway.

6. References

[1] Eduard Forés Ferrer, (2021), “SortAlgorithms”, *GitHub*, https://github.com/eduardfores/SortAlgorithms

[2] Wikipedia contributors, "Bubble sort," *Wikipedia, The Free Encyclopedia,* https://en.wikipedia.org/w/index.php?title=Bubble_sort&oldid=1027858451

[3] Wikipedia contributors, "Cocktail shaker sort," *Wikipedia, The Free Encyclopedia,* https://en.wikipedia.org/w/index.php?title=Cocktail_shaker_sort&oldid=997530496

[4] Wikipedia contributors, "Insertion sort," *Wikipedia, The Free Encyclopedia,* https://en.wikipedia.org/w/index.php?title=Insertion_sort&oldid=1028983026

[5] Wikipedia contributors, "Bucket sort," *Wikipedia, The Free Encyclopedia,* https://en.wikipedia.org/w/index.php?title=Bucket_sort&oldid=1008762131

[6] Wikipedia contributors, "Counting sort," *Wikipedia, The Free Encyclopedia,* https://en.wikipedia.org/w/index.php?title=Counting_sort&oldid=1028705451

[7] Wikipedia contributors, "Merge sort," *Wikipedia, The Free Encyclopedia,* https://en.wikipedia.org/w/index.php?title=Merge_sort&oldid=1029065574

[8] Wikipedia contributors, "Radix sort," *Wikipedia, The Free Encyclopedia,* https://en.wikipedia.org/w/index.php?title=Radix_sort&oldid=1020700798

[10] GeeksforGeeks, “Bubble Sort – GeekforGeeks”, *GeeksforGeeks,* https://www.geeksforgeeks.org/bubble-sort/

[11] GeeksforGeeks, “Cocktail Sort – GeekforGeeks”, *GeeksforGeeks,* https://www.geeksforgeeks.org/cocktail-sort/

[12] GeeksforGeeks, “Insertion Sort – GeekforGeeks”, *GeeksforGeeks,* https://www.geeksforgeeks.org/insertion-sort/

[13] GeeksforGeeks, “Bucket Sort – GeekforGeeks”, *GeeksforGeeks,* https://www.geeksforgeeks.org/bucket-sort-2/

[14] GeeksforGeeks, “Counting Sort – GeekforGeeks”, *GeeksforGeeks,* https://www.geeksforgeeks.org/counting-sort/

[15] GeeksforGeeks, “Merge Sort – GeekforGeeks”, *GeeksforGeeks,* https://www.geeksforgeeks.org/merge-sort/

[16] GeeksforGeeks, “Radix Sort – GeekforGeeks”, *GeeksforGeeks,* https://www.geeksforgeeks.org/radix-sort/