Python programming
Lab. Work n° 5 : Basic sorting 2

Preliminaries : Please remind that teachers can be called to help you on any problem you get. Don't get stuck on an issue for too long.

Exercise n°1 : Basic insertion

1. Create a module `insertion.py` and write a function `insertion(a)` that sort the array a with the insertion sort, and test it through `insertion_main.py`.

As Donald Knuth wrote in its masterpiece (*The Art of Computer Programming, Sorting and Searching, vol. 3*) :

> One of the important families of sorting techniques is based on the "bridge player" method [...] : Before examining record R_j, we assume that the preceding records $R_1, \ldots, R_{j-1}$ have all been sorted ; then we insert R_j into its proper place among the previous records.

(Rephrasing Knuth's description) The basic insertion sort is to compare R_j with $R_{j-1}, R_{j-2}, \ldots$, in turn, until discovering that $R[j]$ is to be inserted between R_p and R_{p+1} ; then we move records $R_{p+1}, \ldots, R_{j-1}$ one space to the right and put R_j into position $p+1$. It is convenient to combine comparison and moving interleaving them.

```
Size? 18
[0, 42, 46, 25, 17, 10, 1, 3, 20, 33, 47, 2, 28, 8, 22, 52, 13, 25]
[0, 1, 2, 3, 8, 10, 13, 17, 20, 22, 25, 25, 28, 33, 42, 46, 47, 52]
```

2. Modify function `insertion_sort` to add some measures : the number of comparisons and the number of moves.

```
Size? 12
[24, 9, 4, 31, 13, 32, 29, 21, 35, 0, 6, 1]
[0, 1, 4, 6, 9, 13, 21, 24, 29, 31, 32, 35]
49 comparisons and 38 moves.
```

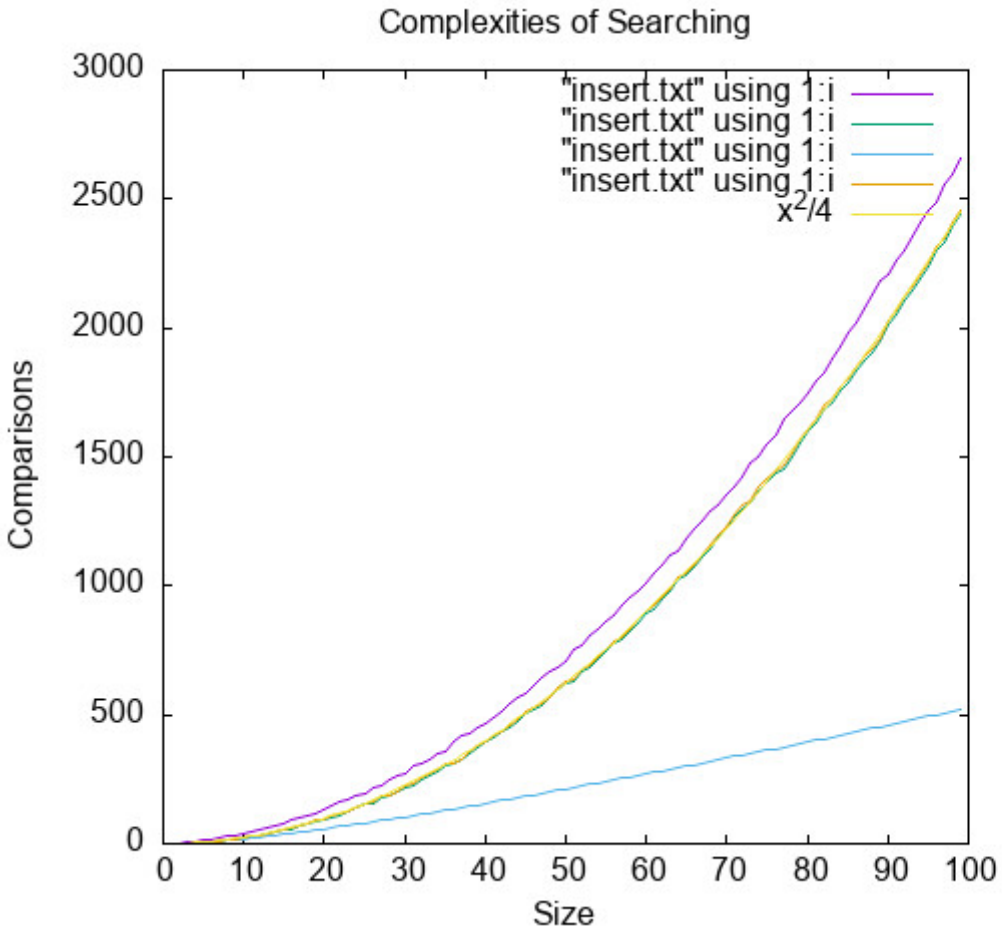
3. Modify the main script so that, e experiences can be launched and the means of the measures printed at the end :

```
Size of the array? 40
Number of experiences? 10000
425.6554 comparisons and 386.6554 moves in the mean.
```

Exercise n°2 : Optimised insertion

1. As we know that in pass i of insertion sort, all elements before i are sorted, we can then use a dichotomic search to find the right place of element to be inserted. Implement it :

```
Number of experiences? 100
Standard insertion sort : 0.0 comparisons and 0.0 moves in the mean size 1.
Dichotomic insertion sort: 0.0 comparisons and 0.0 moves in the mean size 1.
Standard insertion sort : 2.0 comparisons and 0.45 moves in the mean size 2.
Dichotomic insertion sort: 0.0 comparisons and 0.59 moves in the mean size 2.
Standard insertion sort : 4.72 comparisons and 1.4144999999999999 moves in the ▷
    ▷ mean size 3.
Dichotomic insertion sort: 1.0 comparisons and 1.6559 moves in the mean size 3.
Standard insertion sort : 8.1272 comparisons and 2.564145 moves in the mean size ▷
    ▷ 4.
Dichotomic insertion sort: 2.58 comparisons and 3.0065589999999998 moves in the ▷
    ▷ mean size 4.
Standard insertion sort : 11.851272 comparisons and 4.81564145 moves in the mean ▷
    ▷ size 5.
Dichotomic insertion sort: 4.5158 comparisons and 5.390065589999999 moves in the ▷
    ▷ mean size 5.
Standard insertion sort : 16.75851272 comparisons and 6.6981564145 moves in the ▷
    ▷ mean size 6.
Dichotomic insertion sort: 7.0151580000000004 comparisons and 7.333900655899999 ▷
    ▷ moves in the mean size 6.
...
Standard insertion sort : 2613.80656480361 comparisons and 2377.648569757568 ▷
    ▷ moves in the mean size 98.
Dichotomic insertion sort: 513.9573628580285 comparisons and 2392.2220343626886 ▷
    ▷ moves in the mean size 98.
Standard insertion sort : 2653.698065648036 comparisons and 2439.106485697576 ▷
    ▷ moves in the mean size 99.
Dichotomic insertion sort: 520.5795736285803 comparisons and 2453.622220343627 ▷
    ▷ moves in the mean size 99.
```


Complexities of Searching
"insert.txt" using 1:i
"insert.txt" using 1:i
"insert.txt" using 1:i
"insert.txt" using 1:i
x2/4
Comparisons
Size
0
500
1000
1500
2000
2500
3000
0
10
20
30
40
50
60
70
80
90
100