Python programming
Lab. Work n° 6 : Merge sort

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 : Merge sorting

1. Create a module `merge.py` and write a function `merge(l1,l2)` that given two sorted list l_1 and l_2 returns the ordered list of all elements of l_1 and l_2.

Donald Knuth said :

> *Merging* (or *collating*) means the combination of two or more ordered lists into a single ordered list.

```
Size of the first list: 4
[5, 15, 70, 94]
Size of the second list: 6
[32, 39, 52, 81, 94, 95]
[5, 15, 32, 39, 52, 70, 81, 94, 94, 95]
```

2. Implement *merge sort* that can be called through `merge_sort(l)`.

Wikipedia's entry for *merge sort* says :

> Merge sort is a divide-and-conquer algorithm that was invented by John von Neumann in 1945.
>
> [...]
>
> Conceptually, a merge sort works as follows :
>
> - Divide the unsorted list into n sublists, each containing one element (a list of one element is considered sorted).
> - Repeatedly merge sublists to produce new sorted sublists until there is only one sublist remaining. This will be the sorted list.

Try with $n = 2$ first.

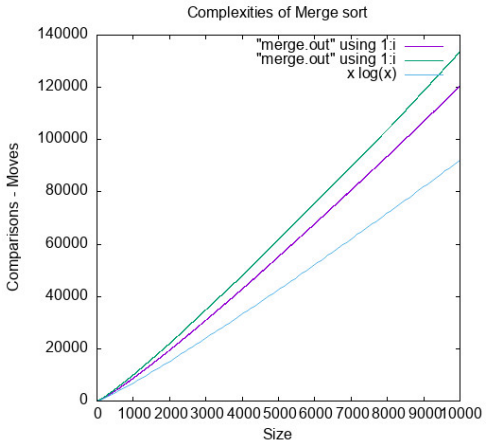
```
Size of the list: 15
[8, 6, 19, 12, 44, 32, 2, 18, 20, 9, 10, 34, 1, 21, 4]
[1, 2, 4, 6, 8, 9, 10, 12, 18, 19, 20, 21, 32, 34, 44]
```

3. Insert instrumentation into the functions to gather the number of comparisons and moves used during the sort. Modify the main script so that, e experiences can be launched and the means of the measures printed at the end :

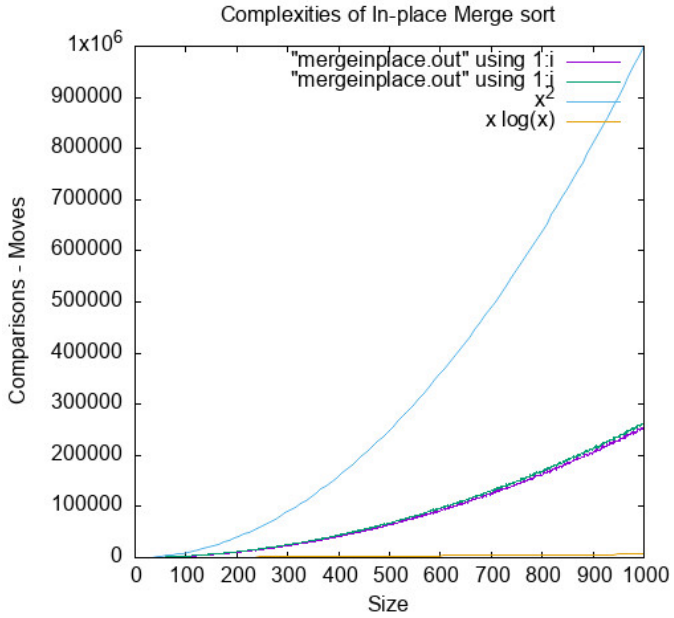
```
Max list size: 100
Number of exp. for each size: 30
Size 2, mean comps 1.0, mean moves 2.0
Size 3, mean comps 2.533333333333333, mean moves 5.0
Size 4, mean comps 4.6, mean moves 8.0
Size 5, mean comps 7.233333333333333, mean moves 12.0
Size 6, mean comps 9.633333333333333, mean moves 16.0
Size 7, mean comps 13.2, mean moves 20.0
```

```
...
Size 97, mean comps 520.5, mean moves 648.0
Size 98, mean comps 528.6666666666666, mean moves 656.0
Size 99, mean comps 535.1666666666666, mean moves 664.0
Size 100, mean comps 541.0333333333333, mean moves 672.0
```

4. Draw the corresponding functions.



5. Implements an in-place merge sort...



6. Optimize the in-place merge sort by using the following trick in the merging phase :

- For every pass, we calculate the *gap* and compare the elements towards the right of the gap.
- Initiate the gap with ceiling value of $n/2$ where n is the combined length of left and right sub-array.
- Every pass, the gap reduces to the ceiling value of $gap/2$.
- Take a pointer i to pass the array.

- Swap the i-th and $(i + gap)$-th elements if $(i + gap)$-th element is smaller than i-th element.
- Stop when $(i + gap)$ reaches n.

As an example (merging in-place 10, 14, 30 and 7, 11, 16, 28) :

Input	10	14	30	7	11	16	28
Gap = 4	**10**	14	30	7	**11**	16	28
	10	**14**	30	7	11	**16**	28
	10	14	**30**	7	11	16	**28**
Gap = 2	**10**	14	**28**	7	11	16	30
	10	**14**	28	**7**	11	16	30
	10	7	**28**	14	**11**	16	30
	10	7	11	**14**	28	**16**	30
	10	7	11	14	**28**	16	**30**
Gap = 1	**10**	**7**	11	14	28	16	30
	7	**10**	**11**	14	28	16	30
	7	10	**11**	**14**	28	16	30
	7	10	11	**14**	**28**	16	30
	7	10	11	14	**28**	**16**	30
	7	10	11	14	16	**28**	**30**
Output	7	10	11	14	16	28	30